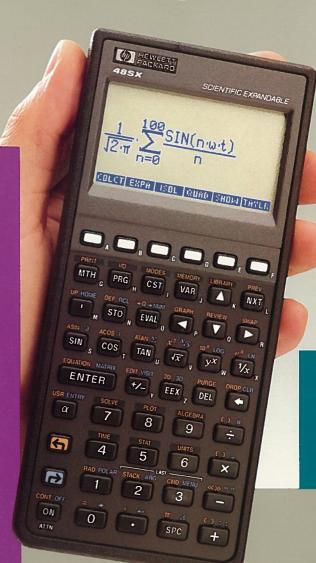


# **HP 48SX**Scientific Expandable



Owner's Manual Volume II

# **HP 48SX Scientific Expandable Calculator**

Owner's Manual Volume II



#### **Notice**

For warranty and regulatory information for this calculator, see pages 673 and 676.

This manual and any examples contained herein are provided "as is" and are subject to change without notice. Hewlett-Packard Company makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard Co. shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein.

 Hewlett-Packard Co. 1990. All rights reserved. Reproduction, adaptation, or translation of this manual is prohibited without prior written permission of Hewlett-Packard Company, except as allowed under the copyright laws.

The programs that control your calculator are copyrighted and all rights are reserved. Reproduction, adaptation, or translation of those programs without prior written permission of Hewlett-Packard Co. is also prohibited.

• Trustees of Columbia University in the City of New York, 1989. Permission is granted to any individual or institution to use, copy, or redistribute Kermit software so long as it is not sold for profit, provided this copyright notice is retained.

Corvallis Division 1000 N.E. Circle Blvd. Corvallis, OR 97330, U.S.A.

### **Printing History**

Edition 1	January 1990	Mfg. No. 00048-90004
Edition 2	April 1990	Mfg. No. 00048-90059
Edition 3	May 1990	Mfg. No. 00048-90062
Edition 4	July 1990	Mfg. No. 00048-90078

# **Contents**

# Part 4: Programming

468	Programming Fundamentals
470	Entering and Executing a Program
470	Entering a Program
472	Executing a Program
472	Editing a Program
473	Using Local Variables
479	Programs That Manipulate Data on the Stack
480	Using Subroutines
483	Single-Step Execution of a Program
484	Single-Step Execution from the Start of the Program
486	Single-Step Execution from the Middle of
	the Program
486	Single-Step Execution of Subroutines
488	Tests and Conditional Structures
	Program Tests
	Comparison Functions
493	Logical Functions
494	Testing Object Types
494	Conditional Structures
	The IFTHENEND Structure
496	The IFTHENELSEEND Structure
	The CASEEND Structure
	Conditional Commands
	The IFT (If-Then-End) Command
500	The IFTE Function
	470 470 472 472 473 479 480 483 484 486 486 486 491 491 493 494 494 494 496 498 499 500

<b>27</b>	501	Loop Structures
	501	Definite Loop Structures
	502	The STARTNEXT Structure
	504	The STARTSTEP Structure
	506	The FORNEXT Structure
	508	The FORSTEP Structure
	510	Indefinite Loop Structures
	510	The DOUNTILEND Structure
	512	The WHILEREPEATEND Structure
	513	Loop Counters (INCR and DECR)
28	515	Flags
	515	Flag Types
	516	Setting, Clearing, and Testing Flags
	518	Recalling and Storing the Flag States
	518	Recalling the Flag States
	518	Storing the Flag States
		salidated from the state
29	519	Interactive Programs
	520	Suspending Program Execution for Data Input
	521	The PROMPT Command
	523	The BEEP Command
	523	The DISP, HALT and FREEZE Commands
	524	The INPUT Command
	531	Labeling Program Output
	532	Using Tagged Objects as Data Output
	533	Using String Commands to Label Data Output
	534	Pausing to Display Data Output
	534	Using Menus in Programs
	534	Displaying a Built-In Menu
	535	Custom Menus in Programs
	539	Building a Temporary Menu
	539	Commands That Return a Key Location
	539	The WAIT Command with Argument 0
	539	The WAIT Command with Argument -1
	540	The KEY Command
	540	Turning the HP 48 Off from a Program

30	541	Error Trapping
	543	The IFERRTHENEND Structure
	544	The IFERRTHENELSEEND Structure
	546	User-Defined Errors
24		Man Barranaina Francis
31	547	More Programming Examples
	548	Fibonacci Numbers
	548	FIB1 (Fibonacci Numbers, Recursive Version)
	550	FIB2 (Fibonacci Numbers, Loop Version)
	551	FIBT (Comparing Program-Execution Time)
	554	Displaying a Binary Integer
	554	PAD (Pad with Leading Spaces)
	555	PRESERVE (Save and Restore Previous Status)
	557	BDISP (Binary Display)
	560	Median of Statistics Data
	561	SORT (Sort a List)
	563	LMED (Median of a List)
	565	MEDIAN (Median of Statistics Data)
	568	Expanding and Collecting Completely
	569	MULTI (Multiple Execution)
	570	EXCO (Expand and Collect Completely)
	572	Finding the Minimum or Maximum Element of
	as till om	an Array
	573	MNX (Finding the Minimum or Maximum Element of an Array—Technique 1)
	576	MNX2 (Finding the Minimum or Maximum Element of an Array—Technique 2)
	579	Verification of Program Arguments
	580	NAMES (Does the List Contain Exactly Two Names?)
	582	VFY (Verify Program Argument)
	585	Bessel Functions
	588	Animation of Successive Taylor's Polynomials
	588	Drawing a Sine Curve and Converting It to a
	555	Graphics Object
	589	Superposition of Successive Taylor's Polynomials
	591	Animation of Taylor's Polynomials

592 Programmatic Use of Statistics and Plotting

597 Animation of a Graphical Image

# Part 5: Printing, Data Transfer, and Plug-Ins

32 602 **Printing** 602 Printing with an HP 82240B Printer **Print Formats** 604 **Basic Printing Commands** 605 Printing a Text String 606 606 Printing a Graphics Object 607 **Double Space Printing** 607 Setting the Delay The HP 48 Character Set 607 Sending Escape Sequences and Control Codes 608 Accumulating Data in the Printer Buffer 608 Printing with an HP 82240A Infrared Printer 609 610 Printing to the Serial Port 611 The PRTPAR Variable 33 612 Transferring Data to and from the HP 48 613 Types of Data You Can Transfer 614 The I/O Menu 616 Local and Server Modes Setting the I/O Parameters 617 617 The SETUP Menu 618 The IOPAR Variable Transferring Data between Two HP 48's 619 Transferring Data between a Computer and the HP 48 621 621 Cable Connection 622 Transferring Data Backing Up All of HP 48 Memory 624 626 Character Translations (TRANSIO) More About File Names 628 629 Errors 629 **ASCII and Binary Transmission Modes** 

	631	Sending Commands to a Server (PKT)
	632	Serial Commands
34	635	Using Plug-in Cards and Libraries
	635	Types of Memory
	636	Installing and Removing Plug-In Cards
	639	RAM Cards
	639	Preparing the Card for Installation
	642	Uses for RAM Cards
	643	Using RAM Cards to Expand User Memory (Merged
		Memory)
	644	Using RAM Cards for Backup (Independent Memory
	645	Backing Up Objects into Independent Memory
	646	Accessing Backup Objects
	647	Backing Up Objects into User Memory (Port 0)
	648	Backing Up All of Memory
	649	Freeing Merged Memory
	651	Using Application Cards and Libraries
	651	Attaching a Library to a Directory
	652	Accessing Library Operations (The LIBRARY
		Menu)
	653	Additional Commands That Access Libraries

# Appendixes and Indexes

656 Calculator Support 656 Answers to Common Questions 660 Environmental Limits 660 When to Replace Batteries 661 Changing Batteries 661 Battery Types 661 Changing Calculator Batteries 663 Changing a RAM Card Battery 665 Testing Calculator Operation 667 Self-Test	A	050	Current Detteries and Comics
660 Answers to Common Questions 660 Environmental Limits 660 When to Replace Batteries 661 Changing Batteries 661 Battery Types 661 Changing Calculator Batteries 663 Changing a RAM Card Battery 665 Testing Calculator Operation	A	656	Support, Batteries, and Service
<ul> <li>660 Environmental Limits</li> <li>660 When to Replace Batteries</li> <li>661 Changing Batteries</li> <li>661 Battery Types</li> <li>661 Changing Calculator Batteries</li> <li>663 Changing a RAM Card Battery</li> <li>665 Testing Calculator Operation</li> </ul>		656	Calculator Support
<ul> <li>When to Replace Batteries</li> <li>Changing Batteries</li> <li>Battery Types</li> <li>Changing Calculator Batteries</li> <li>Changing a RAM Card Battery</li> <li>Testing Calculator Operation</li> </ul>		656	Answers to Common Questions
<ul> <li>661 Changing Batteries</li> <li>661 Battery Types</li> <li>661 Changing Calculator Batteries</li> <li>663 Changing a RAM Card Battery</li> <li>665 Testing Calculator Operation</li> </ul>		660	Environmental Limits
<ul> <li>661 Battery Types</li> <li>661 Changing Calculator Batteries</li> <li>663 Changing a RAM Card Battery</li> <li>665 Testing Calculator Operation</li> </ul>		660	When to Replace Batteries
<ul> <li>661 Changing Calculator Batteries</li> <li>663 Changing a RAM Card Battery</li> <li>665 Testing Calculator Operation</li> </ul>		661	Changing Batteries
<ul><li>Changing a RAM Card Battery</li><li>Testing Calculator Operation</li></ul>		661	
665 Testing Calculator Operation		661	Changing Calculator Batteries
		663	Changing a RAM Card Battery
667 Self-Test		665	Testing Calculator Operation
		667	Self-Test

	669	Port RAM Test
	670	IR Loop-Back Test
	671	Serial Loop-Back Test
	673	Limited One-Year Warranty
	674	If the Calculator Requires Service
	676	Regulatory Information
		a literatura per esta esta a constanti de la c
В	677	Messages
C	694	HP 48 Character Codes
D	697	Menu Numbers
E	699	Listing of HP 48 System Flags
	707	Operation Index
	823	Index

Keyboard Test

# Part 4

# **Programming**

## **Programming Fundamentals**



A program is an object defined by « » delimiters. A program is itself composed of objects and commands whose execution is *delayed* until the program is executed. Because a program is an object, it can be:

- Placed on the stack.
- Stored in a variable.
- Executed repeatedly.
- Executed by another program.

The following example calculates the volume of a sphere, first using keystrokes and then using a program.

**Example: Calculations with Keystrokes and with a Program.** The volume of a sphere of radius *r* is calculated by:

$$V = \frac{4}{3}\pi r^3$$

To do one calculation, you can use the following keystrokes. (Assume you have already placed the radius on the stack.)

3 
$$y^x \leftarrow \pi \times 4 \times 3 \div \rightarrow NUM$$

Each time you press a command key, it is immediately executed, leaving an intermediate result on the stack.

If you want to calculate the volumes of many spheres, you can create a program. The following program assumes the radius is on the stack at the start of program execution:

After keying in the « » delimiters (by pressing ( ), you use the same keystrokes to enter the subsequent objects and commands as you did before. However, the objects and commands that you type are simply listed in the command line—their execution is delayed until you execute the program itself.

Because the program is an object, you can place it on the stack and save it in a variable. To place the program on the stack, press ENTER. To store the program in a variable named VOL, type [] VOL STO. Now you can calculate the volume of any sphere simply by placing the radius on the stack and executing VOL (select the VAR menu and press VOL). You can execute VOL as many times as you want; it acts like a built-in command.

VOL is a program of the simplest form; a series of objects and commands, written in the same order as you would type them from the keyboard. In following chapters, you'll learn about more advanced HP 48 programming features:

- Conditional expressions (chapter 26).
- Looping structures (chapter 27).
- Flags (chapter 28).
- Interactive programs (chapter 29).
- Error trapping (chapter 30).

This chapter covers basic HP 48 programming concepts:

- Entering and executing programs.
- Editing programs.
- Using local variables in programs.
- Stack manipulation of data in programs.

- Using subroutines.
- Single-step execution of programs.

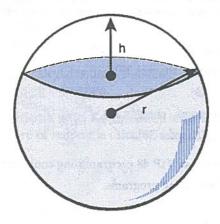
The *Programmer's Reference Manual* for the HP 48 (part number 00048-90054) contains useful programming information, including complete syntax information for all HP 48 commands.

## **Entering and Executing a Program**

#### **Entering a Program**

To define the beginning of a program, press (www.). The PRG annunciator appears, indicating Program-entry mode. In this mode, pressing the key for any command now writes the command's name in the command line. (You can also type the command name into the command line with alpha characters.) Only nonprogrammable operations such as and VAR are executed.

The following program, SPH, calculates the volume of a spherical cap of radius r and height h.



The volume is calculated by  $V = \frac{1}{3} \pi h^2 (3r - h)$ .

In this and following chapters on programming, "stack diagrams" are used as appropriate to show what arguments must be on the stack before a program is executed and what results the program leaves on the stack. Here is the stack diagram for SPH.

Arguments	al ni gazanaci na	Results
2:	2:	
1:	1: volume	

The diagram indicates that SPH takes no arguments from the stack and returns the volume of the spherical cap to level 1. (SPH assumes that you have stored the numerical value for the radius in variable R and the numerical value for the height in variable H.)

Program listings are shown with program steps in the left column and associated comments in the right column. Remember, you can either press the command keys or type in the command names to key in the program. In this first listing, the keystrokes are also shown.

Program:	Keys:	Comments:
«	<b>←</b> (×)	Begins the program.
'1/3	0 1 ⊕ 3	Begins the algebraic expression to calculate the volume.
*π*H^2	× ← π × H y² 2	Multiplies by $\pi h^2$ .
*(3*R-H)'	× ←() 3 × R − H ▶ ▶	Multiplies by $3r - h$ , completing the calculation and ending the expression.
→NUM	→ NUM	Converts $\pi$ to a number.
>		Ends the program.
	(ENTER)	Puts the program on the stack.
	SPH STO	Stores the program in variable SPH.

#### **Executing a Program**

There are several ways to execute SPH:

- Type SPH in the command line, then press ENTER.
- Select the VAR menu, then press SPH.
- If the program or the program name is already in level 1, press [EVAL].

**Example: Executing a Program from the VAR Menu.** Use SPH to calculate the volume of a spherical cap of radius r = 10 mm and height h = 3 mm.

First, store the data in the appropriate variables. Then select the VAR menu and execute the program. The answer is returned to level 1 of the stack.

10 [] R STO 3 [] H STO VAR] SPH



### **Editing a Program**

Follow the same rules to edit a program as you do to edit any other object (see "Displaying Objects For Viewing or Editing" on page 66).

**Example: Editing a Program.** Edit SPH so that it stores the number in level 1 into variable H and the number in level 2 into variable R.

Use the VAR menu and VISIT to call SPH to the command line for editing.

VAR ¹ SPH ► VISIT

Move the cursor past the first program delimiter and insert the new program steps.

P () H ► STOP STO

Save the edited version of SPH in the variable. To verify that the changes were saved, recall SPH to the command line.



No further changes need to be made, so press ATTN to abort the editing session or ENTER to resave the program.

The edited version of SPH now takes two arguments from the stack, the height from level 1 and the radius from level 2.

### **Using Local Variables**

The program SPH in the previous section uses global variables for data storage and recall. There are disadvantages to using global variables in programs:

- After program execution, global variables that you no longer need to use must be purged if you want to clear the VAR menu and free user memory.
- You must explicitly store data in global variables prior to program execution, or have the program execute STO.

In this section, you'll see how *local variables* address the disadvantages of global variables in programs. Local variables are temporary variables created by a program. They exist only while the program is being executed and cannot be used outside the program. They never appear in the VAR menu.

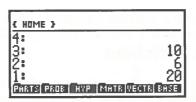
To create local variables, you must use the following sequence of command and objects, called a *local variable structure*:

- **1.** The  $\rightarrow$  command (press  $\rightarrow$   $\rightarrow$ ).
- 2. One or more variable names.
- **3.** A procedure (an algebraic expression or a program) that includes the names. This procedure is called the *defining* procedure.

The structure looks like this:

or

When the  $\rightarrow$  command is executed in a program, n values are taken from the stack and assigned to variables  $name_1$ ,  $name_2$ , ...  $name_n$ . For example, if the stack contains:



#### then:

- $\rightarrow$  a creates local variable a = 20.
- $\Rightarrow$  a b creates local variables a = 6 and b = 20.
- $\rightarrow$  a b c creates local variables a = 10, b = 6, and c = 20.

The defining procedure then uses the local variables to do calculations.

(By convention, this manual uses lowercase names for local variables.)

The following program SPHLV calculates the volume of a spherical cap using local variables. The defining procedure is an algebraic expression.

Arguments	Tradport of bidays	Results	de las considera
2: r 1: h	2: 1: volume		

Program:	Comments:
«	
→ r h	Creates local variables $r$ and $h$ for the radius of the sphere and height of the cap.
'1/3*π*h^2*(3*r-h)'	Expresses the defining procedure. In this program, the defining procedure for the local variable structure is an algebraic expression.
→NUM	Converts $\pi$ to a number.
»	
ENTER 1 SPHLV STO	Stores the program in variable <i>SPHLV</i> .

**Example: Executing a Program That Uses Local Variables.** Use SPHLV to calculate the volume of a spherical cap of radius r=10 mm and height h=3 mm.

Place the data on the stack in the correct order, then select the VAR menu and execute the program.



The preceding program and example demonstrate the advantages of local variable structures:

- The → command stores the value(s) from the stack in the corresponding variable(s) — you do not need to explicitly execute STO.
- Local variables automatically disappear when the defining procedure for which they are created has completed execution. Consequently, local variables do not appear in the VAR menu and occupy user memory only during program execution.
- Local variables exist only within their defining procedure different local variable structures can use the same variable names without conflict.

**Evaluation of Local Names.** Local names are evaluated differently than global names. When a global name is evaluated, the object stored in the corresponding variable is itself evaluated. (You've seen how programs stored in global variables are automatically evaluated when the name is evaluated.)

When a local name is evaluated, the object stored in the corresponding variable is returned to the stack but is *not* evaluated. When a local variable contains a number, the effect is identical to evaluation of a global name, since putting a number on the stack is equivalent to evaluating it. However, if a local variable contains a program, algebraic expression, or global variable name, that object *must be explicitly evaluated* (by executing EVAL) after it is returned to the stack.

**Scope of Local Variables.** Local variables exist *only* in the procedure for which they are defined. The following sample program illustrates the availability of local variables in *nested* defining procedures (procedures within procedures).

Program:	Comments:
«	Starts the outer program.
	For these arbitrary program steps, no local variables are available.
→ a b c	Creates local variables $a, b$ , and $c$ .
«	Starts the defining procedure (a program) for local variables a, b, and c. This procedure is nested in the
	outer program. Local variables $a$ , $b$ , and $c$ are available in this procedure.
a b + c +	
→ d e f	Defines local variables $d$ , $e$ , and $f$ .
mandia - managana mandia mandi	Starts the defining procedure (an algebraic expression) for local variables d, e, and f. This procedure is nested in the defining procedure for local variables a, b, and c. Local variables a, b, c, d, e, and f are

available in this procedure.

a/(d\*e+f)

 $a \in Z$ 

>

×

Ends the defining procedure for local variables d, e, f. Local variables d, e, and f no longer exist.

Local variables a, b, and c remain available.

Ends the defining procedure for local variables a, b, and c. Local variables a, b, and c no longer exist.

For these arbitrary program steps, no local variables are available.

Ends the outer program.

Since local variables a, b, and c already exist when the defining procedure for local variables d, e, and f is executed, they are available for use in that procedure. However, suppose that the defining procedure for local variables d, e, and f calls a program that you previously created and stored in global variable P1.

#### Program:

#### Comments:

«

→ a b c

→ d e f

Defines local variables d, e, and f.

Starts the defining procedure for local variables d, e, and f.

P1+a/(d\*e+f)

The defining procedure executes the program stored in variable P1.

Ends the defining procedure for local variables d, e, and f.

.....

The six local variables are not available in program P1 because they did not exist when you created P1. The objects stored in the local variables are available to program P1 only if you put those objects on the stack as arguments for P1 or store those objects in global variables.

Conversely, program P1 can create its own local variable structure with local variables a, c, and f, for example, without conflicting with the local variables of the same name in the procedure that calls P1.

Programs That Act Like User-Defined Functions. In this chapter you've learned that the defining procedure for a local variable structure can be either an algebraic expression or a program. In chapter 10, you learned that a user-defined function is a program that consists solely of a local variable structure whose defining procedure is an algebraic expression.

A program that begins with a local variable structure whose defining procedure is a program acts like a user-defined function in two ways: It takes numeric or symbolic arguments, and takes those arguments either from the stack or in algebraic syntax. However, it does not have a derivative. (The defining program must, like algebraic defining procedures, return only one result to the stack.)

The advantage of using a program as the defining procedure for a local variable structure is that a program can contain commands not allowed in algebraic expressions. For example, the loop structures described in chapter 27 are not allowed in algebraic expressions. The program BER in chapter 31 calculates a Bessel function approximation to 12-digit accuracy. BER uses a local variable structure whose defining procedure is an RPN program that contains a FOR...STEP structure and a nested IF...THEN...ELSE...END structure. BER is not differentiable, but the example in chapter 31 demonstrates that it can take its arguments either from the stack or in algebraic syntax.

### **Programs That Manipulate Data on the Stack**

The programs SPH (page 471) and SPHLV (page 475) in this chapter use variables for data storage and recall. An alternative programming method manipulates numbers on the stack without storing them in variables. This method usually results in faster program execution time. There are several disadvantages of the stack manipulation method:

- As you write a program, the location of the data on the stack must be tracked. For example, data arguments must be duplicated if used by more than one command.
- A program that manipulates data on the stack is generally harder to read and understand than a program that uses variables.

The following program SPHSTACK uses the stack-manipulation method to calculate the volume of spherical cap. (SPH and SPHLV execute the same calculation.)

Arguments	Results		
2: r 1: h	2: 1: volume		

Program:	Comments:
«	
DUP	Makes a copy of the number in level 1 (the height).
ROT	Rotates the number now in level 3 (the radius) to level 1.
3 *	Multiplies the radius by 3.
SWAP -	Swaps the height into level 1 and subtracts, calculating $3r - h$ .

SWAP SQ *	Swaps the copy of the height into level 1, squares it, and multiplies by $3r - h$ .
π * 3 /	Multiplies by $\pi$ and divides by 3, completing the calculation.
→NUM	Converts $\pi$ to a number.
<b>»</b>	
ENTER P SPHSTACK ISTO	Puts the program on the stack then

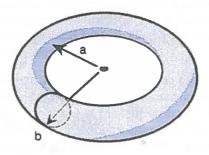
and the same of th

Puts the program on the stack, then stores it in SPHSTACK.

### **Using Subroutines**

Remember that a program is composed of objects and commands that are executed when the program is executed. Because a program is itself an object, it can be used by another program. When program B is used by program A, program A calls program B, and program B is a subroutine in program A.

This section introduces two programs to illustrate the use of subroutines. The first program, *TORSA*, calculates the surface area of a torus of inner radius *a* and outer radius *b*. *TORSA* is used as subroutine in the second program.



The surface area is calculated by:

$$A = \pi^2(b^2 - a^2)$$

Here is the stack diagram and program listing for TORSA.

Arguments		Results	
2: a	2:		
1: b	1: area		

#### 

Program *TORSV* calculates the volume of a torus. It calls *TORSA* to execute part of the calculation.

The formula for the volume of a torus is:

$$V = \frac{1}{4}\pi^{2}(a + b)(b - a)^{2}$$

This equation can be rewritten as:

$$V = \frac{1}{4}\pi^2(b^2 - a^2)(b - a)$$

The quantity  $\pi^2$  ( $b^2 - a^2$ ) in this equation is the surface area of a torus and can be calculated by executing *TORSA*.

Here is a stack diagram for TORSV.

Arguments	Results
2: a	2:
1: b	1: volume

Program:	Comments:
«	
→ a b	Creates local variables $a$ and $b$ .
*	Starts the defining procedure (a program) for the local variable structure.
a b TORSA	Puts the numbers stored in $a$ and $b$ on the stack as arguments for <i>TORSA</i> , then call <i>TORSA</i> to calculate the area $\pi^2(b^2 - a^2)$ .
b a - * 4 /	Completes the volume calculation.
>	Ends the defining procedure.
*	Ends the program.
[ENTER]	Puts the program on the stack.
TORSV STO	Stores the program in TORSV.

TORSV calls program TORSA to execute part of the volume calculation. TORSA is a subroutine in TORSV. In turn, another program can call TORSV.

**Example: Executing a Program That Uses a Subroutine.** Use TORSV to calculate the volume of a torus of inner radius a = 6 inches and outer radius b = 8 inches.

Place the data on the stack according to the stack diagram. Select the VAR menu and execute the program.

6 ENTER 8 VAR TORSY 1: 138.174461616 TORSWITORSHISPHLV H R SPH

### Single-Step Execution of a Program

It's easier to understand how a program works if you execute it step by step, observing the effect of each step. Doing this can help you "debug" your own programs or understand programs written by others.

The operations for single-stepping through a program are contained in the PRG CTRL menu.

#### **Single-Step Operations**

Keys Programmable Command		Description	
(CONT)	CONT Resumes execution of a halted program.		
PRG OTR			
DBUG	C.	Takes as its argument the program or program name in level 1. Starts program execution, then suspends it as if HALT were the first program command.	
888		Executes the next object or command in the suspended program.	
SST		Same as SST except when the next program step is a subroutine. When the next step is a subroutine, single-steps to the first step in that subroutine.	

#### Single-Step Operations (continued)

Keys	Programmable Command	Description
NEXT		Displays the next one or two objects, but does not execute them.
	HALT	Suspends program execution at the location of the HALT command in the program.
KILL	KILL	Cancels all suspended programs.

# Single-Step Execution from the Start of the Program

In many cases, you want to begin single-step execution at the beginning of a program. The general procedure is:

- 1. Put the program or program name in the command line or level 1.
- 2. Press PRG CTRL DEUG Program execution is started, then suspended before execution of the first object or command. The HALT annunciator is displayed in the status area.
- 3. Optional: Press NEXT to display in the status area, but not execute, the next one or two program steps. The display persists until the next keystroke.
- 4. Press SST once to see the first program step displayed in the status area and then executed.
- 5. You can now:
  - Keep pressing SST to display and execute sequential steps.
  - Press NEXT at any time to display but not execute the next one or two program steps.
  - Press ← CONT to continue normal execution.
  - Press KILL to abandon further program execution.

**Example: Single-Step Program Execution.** Execute program TORSV step by step. Use the torus from the previous example (a = 6 inches, b = 8 inches).

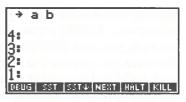
Select the VAR menu and enter the data. Return the program name to the command line. Select the PRG CTRL menu and execute DBUG. The HALT annunciator turns on, indicating that program execution has been started, then suspended.

(VAR)
6 (ENTER) 8 (ENTER)
1 TORSY
PRG CTRL DBUG

E HOME	}	HALT	
4:		-English	WALLEY TO
2			e
08865	551	SST4 N	ST HALT KILL

Execute SST. The first program step is displayed in the status area, then executed.

SST



You can see that the first program step took the two arguments from the stack and stored them in local variables a and b.

Refer to the rules at the beginning of this section. You've executed the first four steps and can now choose one of the four alternatives described in step 5. For this example, continue single-step execution until the HALT annunciator disappears. Watch the stack and status area as you single-step through the program.

SST ... SST

1: 138.174461616

# **Single-Step Execution from the Middle of the Program**

You may want to start single-step execution at some point in the program other than the first step. To do so:

- 1. Insert the HALT command in the program. Place it where you want to begin single-step execution.
- 2. Execute the program. When the HALT command is executed, the program stops and the HALT annunciator is displayed.
- 3. Follow steps 3-5 on page 484.
- **4.** When you want the program to run normally again, remove the HALT command from the program.

#### **Single-Step Execution of Subroutines**

SST executes the next step in a program. If the next step is a subroutine, SST executes that subroutine in one step. In the previous example, you used SST to execute subroutine TORSA in one step. However, you may want to single-step through a subroutine, executing each individual step rather than the program as a whole. To do so, use the SST+ operation. SST+ works just like SST, except when the next program step is a subroutine. In this case, SST+ single-steps to the first step in the subroutine.

**Example: Single-Step Execution of a Subroutine.** Execute program TORSV step by step to calculate the volume of a torus of radii a = 10 inches and b = 20 inches. When you reach subroutine TORSA, execute it step by step.

Select the VAR menu and key in the data. Return the program name to the command line, select the PRG CTRL menu, and execute DBUG. Execute the first four steps of the program, then check the next step.

VAR

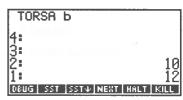
10 ENTER 12

TORSV

PRG CTRL DBUG

SST+ (or SST ) 4 times

NEXT



The next step is TORSA. If you now execute SST, TORSA will be executed. Since you want to single-step through TORSA, execute SST. Then verify that you are now at the first step of TORSA, not the next step of TORSV.

SST↓ NEXT



Execute SST or SST+ repeatedly to single step through the remainder of the program, or at any time, press (CONT) to resume program execution.

### **Tests and Conditional Structures**



This chapter describes commands and program structures that, used together, let programs ask questions and make decisions:

- Comparison functions and logical functions let a program test whether or not a specified condition exists.
- Program structures called conditional structures use test results to make decisions.

**Example: Tests and Conditional Structures.** The program in this example uses a test inside a conditional structure to execute the following task:

"If the two numbers on the stack have the same value, drop one of the numbers from the stack and store the other in variable V1. If, however, the numbers are not equal, store the number from level 1 in V1 and the number from level 2 in V2."

Program: Comments:

« Starts the program.

DUP2 Copies the numbers in levels 1 and 2.

IF Starts the test clause of the

conditional structure.

SAME Tests if the numbers have the same

value.

THEN Ends the test clause and starts the

true clause of the conditional

structure. The true clause is executed

only if the test is true.

DROP If the test is true (if the numbers are 'V1' STO the same), then drops one of the

numbers from the stack and stores

the remaining number in V1.

ELSE Starts the false clause of the

conditional structure. The false clause is executed only if the test is

false.

'V1' STO If the test is false, (if the numbers 'V2' STO are not the same), then stores the

are not the same), then stores the level 1 number in V1 and the level 2

number in V2.

END Ends the conditional structure.

» Ends the program.

ENTER TST STO Puts the program on the stack and

stores it in TST.

Enter the numbers 26 and 52, then execute TST to compare their values.

26 ENTER 52 VAR TST



Since the two number were not equal, the VAR menu now contains two new variables V1 and V2. You can verify that the variables contain the numbers you entered by pressing both menu keys.

### **Program Tests**

A test is an algebraic or a command sequence that returns a test result to the stack. A test result is either a 1—which means the test was true, or a  $\emptyset$ —which means the test was false. For example, ' $\times$ <Y' is a test. The same test could be executed as a command sequence: X Y <. In either case, if X contains 5 and Y contains 10, then the test is true, and 1 is returned to the stack. Conditional structures (discussed later in the chapter) use a test result to determine which clause of the structure to execute.

The commands used in tests can be categorized as follows:

- **■** Comparison functions.
- Logical functions.
- Flag-testing commands. Flags and flag testing commands are discussed in chapter 28, "Flags."

These commands are located in the PRG TEST menu (press PRG TEST).

#### **Comparison Functions**

Comparison functions compare two objects.

#### **Comparison Functions**

Keys	Programmable Command	Description
PRG TE	ST (pages 1 and 2	2):
<	<	Less than.
<b>&gt;</b>	>	Greater than.
<b>4</b>	≤	Less than or equal to.
7	≥	Greater than or equal to.
==		Tests equality of two objects. For algebraics or names, returns an expression that can be evaluated to produce a test result based on numerical values.
	<i>≠</i>	Not equal. Like = =, but returns the opposite test result.
SAME	SAME	Like = =, but does not allow a comparison between the numerical value of an algebraic (or name) and a number.

<, >,  $\leq$  and  $\geq$  compare two real numbers, two binary integers, or two strings returning 1 (true) or  $\emptyset$  (false) based on the comparison. The order of the comparison is *level 2 test level 1*, where test is the comparison function. For example, if 6 is stored in X,  $\times$  5 < removes 6 and 5 from the stack and returns  $\emptyset$ . If one object is an algebraic (or name) and the other object is an algebraic (or name) or a number, <, >,  $\leq$ , and  $\geq$  return an expression that must be evaluated to return a test result. For strings, "less than" means alphabetically previous. For example, "AAA" is less than "AAB".

- = = takes two objects from the stack and:
  - If either object is not an algebraic or a name, returns 1 if the two objects are the same type and have the same value, or ② otherwise. Lists and programs are considered to have the same value if the objects they contain are identical.
  - If one object is an algebraic (or name) and the other object is an algebraic (or name) or a number, returns an expression that must be evaluated to return a test result.

(Note that = = is used for comparisons, while = separates two sides of an equation.)

# works just like = =, except that the test results are opposite.

SAME returns 1 (true) if two objects identical. For example, 'X+3'  $\stackrel{?}{\sim}$  SAME returns 0 regardless of the value of X because the algebraic 'X+3' is not identical to the real number 4. For all object types other than algebraics and names, SAME works just like = =.

Using Comparison Functions in Algebraics. Comparison functions (except SAME) can be used in algebraics as *infix* functions. For example, if 6 is stored in X, 'X<5'  $\rightarrow NUM$  returns O.

#### **Logical Functions**

Logical functions return a test result based on the outcomes of two previously executed tests. Note that these four functions interpret any non-zero argument as a true result.

#### **Logical Functions**

Keys	Programmable Command	Description
PRG TE	EST (page 1):	
AND	AND	Returns 1 (true) if both arguments are true.
(#)?	OR	Returns 1 (true) if either or both arguments are true.
XOR	XOR	Returns 1 (true) if either, but not both, arguments are true.
NOT	NOT	Returns 1 (true) if the argument is Ø (false); otherwise, returns Ø (false).

AND, OR, and XOR are used to combine two test results. For example, if 4 is stored in Y, Y 8 < 5 AND returns 1. First, Y 8 < returns 1 to the stack. AND removes 1 and 5 from the stack, interpreting both as true results, and returns 1 to the stack.

**Using Logical Functions in Algebraics.** AND, OR, and XOR can be used as *infix* functions in algebraics. For example, '3<5 XOR 4>7' →NUM returns 1.

NOT can be used as a *prefix* function in algebraics. For example, 'NOT  $Z \le 4$ '  $\Rightarrow$ NUM returns  $\emptyset$  if Z = 2.

## **Testing Object Types**

The TYPE command (PRG TEST TYPE) takes any object as its argument and returns the number that identifies that object type. The table on page 97 in chapter 4 lists the HP 48 objects and their corresponding type number.

## **Conditional Structures**

The HP 48 conditional structures let a program make a decision based on the result of a test or tests. Conditional structures are built with commands that work only when used in proper combination with each other. These commands are contained in the PRG BRCH menu (PRG BRCH).

The conditional structures are:

- IF...THEN...END.
- IF...THEN...ELSE...END.
- CASE...END.

### The IF...THEN...END Structure

IF...THEN...END executes a sequence of commands only if a test evaluates to true. The syntax is:

IF test-clause THEN true-clause END

The test-clause can be a command sequence (for example,  $\exists \exists$ ) or an algebraic (for example,  $\exists \exists \exists$ ). If the test-clause is an algebraic, it is automatically evaluated to a number ( $\rightarrow$ NUM or EVAL isn't necessary).

As a typing aid, press IF to key in:

IF THEN END **Example 1: IF...THEN...END.** Both programs below test the value in level 1. If the value is positive it is made negative. The first program uses a command sequence as the test-clause:

#### « DUP IF 0 > THEN NEG END »

The value on the stack must be duplicated because the > command removes two arguments from the stack (the copy of the value made by DUP, and 0).

The next version uses an algebraic as the test clause:

**Example 2: IF...THEN...END.** This program multiplies two numbers together if both are non-zero.

Program:	Comments:	
«		
→ x y	Creates local variables x and y containing the two numbers from the stack.	
«		
IF	Starts the test-clause.	
'×≠0'	Tests one of the numbers and leaves a test result on the stack.	
'y≠0'	Tests the other number, leaving another test result on the stack.	
AND	Tests whether both tests were true.	
THEN	Ends the test-clause, starts the true-clause.	
ху*	If AND returns true, multiplies the two numbers together.	

END

» »

The following program accomplishes the same task as the previous program:

The test-clause 'x AND y' returns "true" if both numbers are non-zero.

**How IF...THEN...END Works.** IF begins the test-clause, which leaves a test result on the stack. THEN removes the test result from the stack. If the value is non-zero, the true-clause is executed. Otherwise, program execution resumes following END.

## The IF...THEN...ELSE...END Structure

IF...THEN...ELSE...END executes one sequence of commands if a test is true, and another sequence of commands if that test is false. The syntax is:

IF test-clause THEN true-clause ELSE false-clause END

If the test-clause is an algebraic, it is automatically evaluated to a number (—NUM or EVAL isn't necessary).

As a typing aid, press F IF to key in:

IF THEN ELSE END **Example 1: IF...THEN...ELSE...END.** The following program takes a value x from the stack and calculates  $\sin x/x$ . At x = 0 the division would error, so the program returns the limit value 1 in this case:

« + x « IF 'x ≠ 0' THEN x SIN x / ELSE 1 END » »

**Example 2: IF...THEN...ELSE...END.** This program, like example 2 for IF...THEN...END, multiplies two numbers together if they are both non-zero. However, the program returns the string "ZERO" if either value is 0.

Program:	Comments:
«	
→ n1 n2	Stores the values from levels 1 and 2 in local variables.
«	Starts the defining procedure for the local variable structure.
IF	Starts the test clause.
'n1≠0 AND n2≠0'	Tests n1 and n2.
THEN	If both numbers are non-zero
n1 n2 *	multiplies the two values.
ELSE	If both numbers are not non-zero
"ZERO"	returns the string ZERO.
END	Ends the conditional.
*	Ends the defining procedure.

How IF...THEN...ELSE...END Works. IF begins the test-clause, which leaves a test result on the stack. THEN removes the test result from the stack. If the value is non-zero, the true-clause is executed. Otherwise, the false-clause is executed. After the appropriate clause is executed, execution resumes following END.

#### The CASE...END Structure

The CASE...END structure lets you execute a series of cases (tests). The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE...END structure. Optionally, you can include after the last test a default clause that is executed if all the tests evaluate to false.

The CASE...END structure has the syntax:

CASE

test-clause<sub>1</sub> THEN true-clause<sub>1</sub> END test-clause<sub>2</sub> THEN true-clause<sub>2</sub> END

test-clause, THEN true-clause, END default-clause (optional)

END

CASE

THEN

**END** 

END

and CASE to key in:

THEN

END

**Example: The CASE...END Structure.** The following program stores the level 1 argument in a variable if the argument is a string, list, or program.

Program:	Comments:
«	
→ y	Stores the argument in local variable y.
«	Starts the defining procedure.
CASE	Starts the case structure.
y TYPE 2 SAME THEN y 'STR' STO END	Case 1: If the argument is a string, stores it in STR.
y TYPE 5 SAME THEN y 'LIST' STO END	Case 2: If the argument is a list, stores it in <i>LIST</i> .
y TYPE 8 SAME THEN y 'PROG' STO END	Case 3: If the argument is a program, stores it in <i>PROG</i> .
END	Ends the case structure.
>	Ends the defining procedure.

**How CASE...END Works.** When CASE is executed, test-clause<sub>1</sub> is evaluated. If the test is true, true-clause<sub>1</sub> is executed, and execution skips to END. If test-clause<sub>1</sub> is false, execution proceeds to test-clause<sub>2</sub>. Execution within the CASE structure continues until a true-clause is executed, or until all the test-clauses evaluate to false. Optionally, a default clause can be included. In this case, the default-clause is executed if all the test-clauses evaluate to false.

## **Conditional Commands**

The IF...THEN...END and IF...THEN...ELSE structures are useful for situations where the true-clause and false-clause are sequences of commands and objects. Two commands, IFT (If...Then) and IFTE (If...Then...Else), let you easily execute the same decision-making process if the true- and false-clauses are each a single command or object.

## The IFT (If-Then-End) Command

The IFT command takes two arguments: a test result in level 2 and an object in level 1 (the "true clause"). The object in level 1 is executed if the test result is true.

**Example: The IFT Command.** The following program removes a number from the stack and displays POSITIVE if the number is positive.

« Ø > "POSITIVE" IFT »

#### The IFTE Function

The IFTE function takes three arguments: a test result in level 3, and objects in levels 2 and 1. The level-2 object (the "true-clause") is executed if the test result is true. Otherwise, the level-1 object (the "false-clause") is executed.

**Example: The IFTE Command.** This program takes a value from level 1 and displays POSITIVE if it is positive or zero, and NEGATIVE otherwise:

« Ø ≥ "POSITIVE" "NEGATIVE" IFTE »

**Using IFTE in Algebraics.** The IFTE function can also be used as a function in algebraics. It has the syntax:

IFTE(test, true-clause, false-clause)

**Example: The IFTE Function.** This program is a user-defined function that takes a number (x) from the stack and calculates  $\sin(x)/x$  if x is non-zero. If x is 0, the program returns 1:

« → x 'IFTE(x≠0,SIN(x)/x,1)' »

# **Loop Structures**



Loop structures execute a part of a program repeatedly. There are two fundamental types of loops:

- For a *definite loop*, the program specifies in advance how many times the loop clause will be executed.
- In an *indefinite loop*, the program uses a test to determine whether to execute the loop-clause again.

Like the conditional structures described in chapter 26, looping structures are built with commands that work only when used in proper combination with each other. These commands are contained in the PRG BRCH menu (PRG BRCH).

# **Definite Loop Structures**

There are two definite loop structures. Each has two variations:

- START...NEXT and START...STEP.
- FOR...NEXT and FOR...STEP.

## The START...NEXT Structure

START...NEXT executes a portion of a program a specified number of times. The syntax is:

start finish START loop-clause NEXT

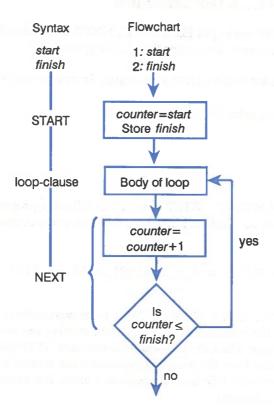
As a typing aid, press START to key in:

START NEXT

**Example: A START...NEXT Loop.** The following program creates a list containing ten copies of the string "ABC":

« 1 10 START "ABC" NEXT 10 →LIST »

How START...NEXT Works. START takes two numbers (start and finish) from the stack and stores them as the starting and ending values for a loop counter. Then, the loop-clause is executed. NEXT increments the counter by 1 and tests to see if its value is less than or equal to finish. If so, the loop-clause is executed again.



Notice that the loop-clause is always executed at least once.

### The START...STEP Structure

START...STEP works just like START...NEXT, except that it lets you specify an increment value other than 1. The syntax is:

start finish START loop-clause increment STEP

As a typing aid, press FART to key in:

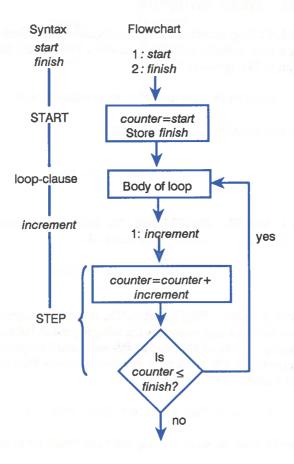
START STEP

**Example: A START...STEP Loop.** The following program takes a number x from the stack and calculates the square of that number x/3 times:

« DUP → x « x 1 START x SQ -3 STEP »

How START...STEP Works. START takes two numbers (start and finish) from the stack and stores them as the starting and ending values of the loop counter. Then, the loop-clause is executed. STEP takes the increment value from the stack and increments the counter by that value. If the argument of STEP is an algebraic or a name, it is automatically evaluated to a number.

The increment value can be positive or negative. If it is positive, the loop is executed again when the counter is less than or equal to *final*. If the increment value is negative, the loop is executed when the counter is greater than or equal to *final*. In the following flowchart, the increment value is positive.



### The FOR...NEXT Structure

A FOR...NEXT loop executes a program segment a specified number of times using a local variable as the loop counter. You can use this variable within the loop. The syntax is:

start finish FOR counter loop-clause NEXT

As a typing aid, press FOR to key in:

FOR NEXT

**Example 1: A FOR...NEXT Loop.** The following program places the squares of the integers 1 through 5 on the stack:

« 1 5 FOR j j SQ NEXT »

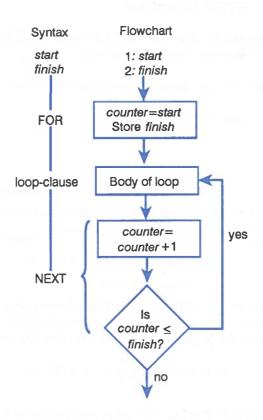
**Example 2: A FOR...NEXT Loop.** The following program takes the value x from the stack and computes the integer powers i of x. For example, when x = 12 and start and finish are 3 and 5 respectively, the program returns  $12^3$ ,  $12^4$ , and  $12^5$ . It requires as inputs start and finish in levels 3 and 2, and x in level 1:

« → x « FOR n 'x^n' EVAL NEXT » »

 $\rightarrow$  x removes x from the stack, leaving start and finish there as arguments for FOR.

**How FOR...NEXT Works.** FOR takes *start* and *finish* from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Then, the loop-clause is executed; *counter* can appear within the loop clause. NEXT increments *counter* by one, and then tests whether *counter* is less than or equal to *finish*. If so, the *loop-clause* is repeated (with the new value of *counter*).

When the loop is exited, counter is purged.



### The FOR...STEP Structure

FOR...STEP works just like FOR...NEXT, except that it lets you specify an increment value other than 1. The syntax is:

start finish FOR counter loop-clause increment STEP

As a typing aid, press FOR to key in:

FOR STEP

**Example 1: A FOR...STEP Loop.** The following program places the squares of the integers 1, 3, 5, 7, and 9 on the stack:

« 1 9 FOR x x SQ 2 STEP »

**Example 2: A FOR...STEP Loop.** The following program takes n from the stack, and returns the series of numbers 1, 2, 4, 8, 16, ... n. If n isn't in the series, the program stops at the last value less than n:

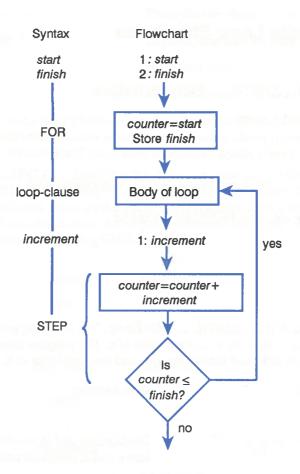
« 1 SWAP FOR n n n STEP »

How FOR...STEP Works. FOR takes start and finish from the stack as the beginning and ending values for the loop counter, then creates the local variable counter as a loop counter. Next, the loop-clause is executed; counter can appear within the loop clause. STEP takes the increment value from the stack and increments counter by that value.

The increment value can be positive or negative. If the increment is positive, the loop is executed again when *counter* is less than or equal to *final*. If the increment is negative, the loop is executed when *counter* is greater than or equal to *final*.

When the loop is exited, counter is purged.

(In the following flowchart, the increment value is positive.)



# **Indefinite Loop Structures**

### The DO...UNTIL...END Structure

DO...UNTIL...END... executes a loop repeatedly until a test returns a true (non-zero) result. Since the test-clause is executed after the loop-clause, the loop is always executed at least once. The syntax is:

DO loop-clause UNTIL test-clause END

As a typing aid, press 🕤 DO to key in:

DO UNTIL END

**Example: A DO...UNTIL...END Loop.** The following program calculates n + 2n + 3n + ... for a value of n. The program stops when the sum exceeds 1000, and returns the sum and the coefficient of n.

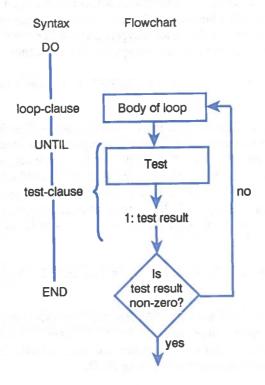
Program:	Comments:
«	
DUP 1 → n s c 4	Duplicates $n$ and stores the value into $n$ and $s$ ; initializes counter $c$ to 1.
«	Starts the defining procedure, in this case a program, for the local variable structure.
DO	Starts the loop-clause.
'c' INCR	Increments the counter by 1. (INCR is discussed on page 513.)
n * 's' STO+	Calculates $c \times n$ , and adds the product to $s$ .

UNTIL Starts the test clause. s 1000 > Repeats loop until s > 1000.

END Ends the test-clause.  $s \subset Puts s \text{ and } c \text{ on the stack.}$   $s \subset Puts s \text{ the defining procedure.}$ 

×

How DO...UNTIL...END Works. DO starts execution of the loop-clause. UNTIL ends the loop clause and begins the test-clause. The test-clause leaves a test result on the stack. END removes the test result from the stack. If its value is zero, the loop-clause is executed again; otherwise, execution resumes following END.



## The WHILE...REPEAT...END Structure

WHILE...REPEAT...END repeatedly evaluates a test and executes a loop-clause if the test is true. Since the test-clause occurs before the loop-clause, the loop-clause is never executed if the test is initially false. The syntax is:

WHILE test-clause REPERT loop-clause END

As a typing aid, press HHILE to key in:

WHILE REPEAT END

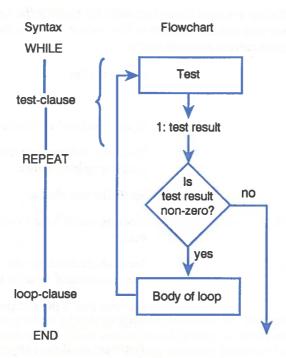
**Example 1: A WHILE...REPEAT...END Loop.** The following program starts with a number on the stack, and repeatedly performs a division by 2 as long as the result is evenly divisible. For example, starting with the number 24, the program computes 12, then 6, then 3:

« WHILE DUP 2 MOD 0 == REPEAT 2 / DUP END DROP »

**Example 2: A WHILE...REPEAT...END Loop.** The following program takes any number of vectors or arrays from the stack and adds them to the statistics matrix. (The vectors and arrays must have the same number of columns.) WHILE...REPEAT...END is used instead of DO...UNTIL...END because the test must be done *before* the addition. (If *only* vectors or arrays with the same number of columns are on the stack, the program errors after the last vector or array is added to the statistics matrix.)

« WHILE DUP TYPE 3 == REPEAT X+ END »

How WHILE...REPEAT...END Works. The test-clause is executed and returns a test result to the stack. REPEAT takes the value from the stack. If the value is non-zero, execution continues with the loop-clause; otherwise, execution resumes following END.



## **Loop Counters (INCR and DECR)**

The INCR (increment) command (MEMORY INCR) takes a global or local variable name as its argument. The variable must contain a real number. The command:

- Returns the new value of the variable.
- Increments by 1 the value stored in the variable.

For example, if c contains the value 5, 'c' INCR returns 6 to the stack and stores 6 in c.

The DECR (decrement) command is analogous to INCR, except that it subtracts 1 from the specified variable.

**Example: Using a Loop Counter with an Indefinite Loop.** The following program takes a maximum of five vectors from the stack and adds them to the current statistics matrix.

Program:	Comments:
" 0 → c	Stores 0 in local variable c.
«	Starts the defining procedure for the local variable structure.
WHILE	Starts the test clause.
DUP TYPE 3 ==	Returns true if level 1 contains a vector.
'c' INCR	Increments the value in c and puts the incremented value in level 1.
5	Returns true if the incremented value of $c \le 5$ .
AND	Returns true if the two previous test results are true.
REPEAT	
Σ+	Adds the vector to $\Sigma DAT$ .
END	Ends the WHILEREPEAT structure.
»	Ends the defining procedure.

# **Flags**



Flags are an important programming tool in the HP 48. You can think of a flag as a switch that is either on (set) or off (clear). A program can test a flag's state within a conditional or looping structure (described in the previous chapters) to make a decision. Since flags have unique meanings for the calculator, flag tests expand a program's decision-making capabilities beyond that available with comparison and logical functions.

# Flag Types

There are two types of flags in the HP 48: system flags, numbered -1 through -64; and user flags, numbered 1 through 64. System flags have a predefined meaning for the calculator. For example, system flag -40 controls the clock display—when this flag is clear (the default state), the clock is displayed only when the TIME menu is selected; when this flag is set, the clock is displayed at all times. (Actually, when you press CEK in the MODES menu, you set or clear flag -40.) Appendix E lists the 64 system flags and their definitions.

User flags are not used by any built-in operations; what they mean depends entirely on how you define them. When you set a user flag 1 through 5, the corresponding annunciator is activated. (Note that plug-in cards, described in chapter 34, may affect the settings of user-flags 31—64.)

## Setting, Clearing, and Testing Flags

The following commands take as their argument a flag number — an integer 1 through 64 (for user flags), or -1 through -64 (for system flags).

Flag Commands

Keys	Programmable Command	Description
PRG TE	ST (page 3) (or	MODES pages 2 and 3):
- 1: E	SF	Sets the flag.
BE.	CF	Clears the flag.
FS?	FS?	Returns true (1) if the flag is set, or false (0) if the flag is clear.
FC?	FC?	Returns true (1) if the flag is clear, or false (0) if the flag is set.
FS?C	FS?C	Tests the flag (returns true if the flag is set), then clears the flag.
FC?C	FC?C	Tests the flag (returns true if the flag is clear), then clears the flag.

**Example: Testing a System Flag.** The following program sets an alarm for June 6, 1991 at 5:05 PM. It first tests the status of system flag -42 (the Date Format flag) in a conditional structure and then supplies the alarm date in the current date format, based on the test result.

Program:	Comments:
«	
IF -42 FC?	Tests the status of flag -42, the Date Format flag.
THEN 6.151991	If flag -42 is clear, supplies the date in month/day/year format.
ELSE 15.061991	If flag -42 is set, supplies the date in day.month.year format.
END	Ends the conditional.
17.05 "TEST COMPLETE" 3 →LIST STOALARM	Completes the set-alarm command sequence. (17.05 is the alarm time and "TEST COMPLETE" is the alarm message.)

**Example: User Flags in Programs.** The following program returns either the fractional or integer part of the level 1 argument, depending on the state of user flag 10.

**Comments:** 

Program:

. rogram.	
«	
IF	Starts the conditional.
10 FS?	Tests the status of user flag 10.
THEN	If flag 10 is set
IP	returns the integer part.
ELSE	If flag 10 is clear
FP	returns the fractional part.
END	Ends the conditional.

Before you execute this program, you set flag 10 if you want to return the integer part of the argument, or you clear flag 10 if you want to return the fractional part of the argument. Flag 10 is defined to have a unique meaning in the program; its status determines which part of the level 1 argument is returned to the stack.

## **Recalling and Storing the Flag States**

The RCLF (recall flag status) and STOF (store flag status) commands let you recall and then store the status of the HP 48 flags. The commands let a program that alters the status of a flag or flags during execution preserve the pre-program-execution flag status.

## **Recalling the Flag States**

RCLF returns a list containing two 64-bit binary integers that represent the current status of the system flags and user flags respectively:

The rightmost (least significant) bits of  $\#n_s$  and  $\#n_u$  represent the states of system flag -1 and user flag +1 respectively.

## **Storing the Flag States**

STOF sets the current states of the system flags, or the states of both the system and user flags. It takes as its argument either:

- A single binary integer  $(\#n_s)$ , in which case only the corresponding system flags are set or cleared.
- A list containing two binary integers ( $\{ \#n_s \#n_u \}$ ), in which case the corresponding system and user flags are set or cleared.

A bit with value 1 sets the corresponding flag; a bit with value 0 clears the corresponding flag. The rightmost (least significant) bits of  $\#n_s$  and  $\#n_u$  set the states of system flag -1 and user flag +1 respectively.

The program *PRESERVE* on page 555 in chapter 31 uses RCLF and STOF.

# **Interactive Programs**



Simple programs like those in chapter 25 use data that is supplied before program execution and return results as unlabeled numbers. Such programs may be difficult to use, particularly if you are not the program author. You must know what arguments to enter on the stack and in what order to enter them, and you must know how to interpret the results returned to the stack.

Interactive programs do any of the following:

- Stop during execution to prompt you for data.
- Display program results with explanatory messages or tags.
- Stop during execution so that you can make a choice about how you want the program to proceed.

# **Suspending Program Execution for Data Input**

## **Data Input Commands**

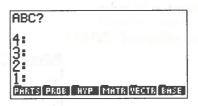
Keys	Programmable Command	Description
(CONT)	CONT	Restarts a halted program.
PRG CTR	(pages 1, 2 and 3	3):
	HALT	Halts program execution.
	INPUT	Suspends program execution for data input. Prevents stack operations while the program is paused.
	PROMPT	Halts program execution for data input.
(P) (S)2	DISP	Displays an object in the specified line of the display.
MASACT.	WAIT	Suspends program execution for <i>x</i> seconds, where <i>x</i> is a number from level 1.
KEY	KEY	Returns a test result to level 1 and, if a key is pressed, the location of that key.
	BEEP	Sounds a beep at a specified frequency for a specified duration.
PRG DSPL (page 4):		
CLLCD	CLLCD	Blanks the display.
EREZ	FREEZE	"Freezes" a specified area of the display so that it is not updated until a key press.

### The PROMPT Command

PROMPT takes a string argument from level 1, displays the string (without the "delimiters) in the status area, and halts program execution. Calculator control is returned to the keyboard. Program execution is resumed by executing CONT. For example, when you execute the program segment:

« "ABC?" PROMPT »

the display looks like this:



The message is displayed until you press **ENTER** or **ATTN** or until you update the status area (for example, by pressing **REVIEW**).

The following program, *TPROMPT*, prompts you for the dimensions of a torus, then calls program *TORSA* (chapter 25, page 481) to calculate its surface area. You don't have to enter data on the stack prior to program execution.

Arguments		Results
1:	1: area	

Program:

**Comments:** 

\*

"ENTER a, b IN ORDER: "

Puts the prompting string on the stack.

PROMPT

Displays the string in the status area, halts program execution, and returns

calculator control to the keyboard.

TORSA

Executes TORSA, using the justentered stack arguments.

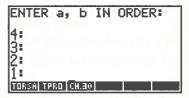
I TPROMPT STO ENTER

Stores the program in TPROMPT.

**Example: Prompting for Data Input in a Program.** Execute TPROMPT to calculate the volume of a torus with inner radius a = 8inches and outer radius b = 10 inches.

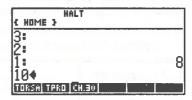
Select the VAR menu and start TPROMPT.

VAR TERO



The program prompts you for data. Enter the inner and outer radii. Note that after you press ENTER, the prompt message is cleared from the status area.

8 [ENTER] 10



Continue the program.

(CONT)



The answer is returned to level 1 of the stack.

Note that when program execution is suspended by PROMPT, you can execute calculator operations just as you did before you started the program. Suppose the outer radius b of the torus in the previous example is measured as 0.83 feet. You can convert that value to inches while the program is suspended for data input by pressing .83 [ENTER] 12 [X].

### The BEEP Command

The BEEP command lets you enhance an interactive program with audible prompting. BEEP takes two arguments from the stack: the tone frequency from level 2 and the tone duration from level 1. The following edited version of *TPROMPT* sounds a 440-hertz, one-half-second tone at the prompt for data input.

#### Program:

#### Comments:

«

"ENTER a, b IN ORDER:"

440 .5 BEEP

Sounds a tone to audibly supplement the prompt for data input.

PROMPT

TORSA

\*

## The DISP, HALT and FREEZE Commands

DISP, HALT, and FREEZE can be used together to prompt for data input:

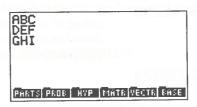
- DISP displays an object in a specified line of the display. DISP takes two arguments from the stack: an object from level 2, and a display-line number 1 through 7 from level 1. To facilitate the display of messages, DISP displays *string* objects without the surrounding "delimiters.
  - Note that the display created by DISP persists only as long as the program continues execution. When the program ends, or when it is suspended by the HALT command, the calculator returns to the normal stack environment, and the display is automatically updated.
- FREEZE "freezes" one or more display areas so they are not updated until a *key press*. Argument n in level 1 is the sum of the value codes for the areas to be frozen. The value codes are: 1 for the status area; 2 for the stack/command line area; 4 for the menu area.

■ HALT suspends program execution at the location of the HALT command and turns on the HALT annunciator. Calculator control is returned to the keyboard for normal operations. Program execution is resumed by executing CONT (or SST).

For example, when you execute the following program:

« "ABC=DEF=GHI" CLLCD 1 DISP 3 FREEZE HALT >>

the display looks like this:



(The ■ in the previous program is the calculator's representation for the 

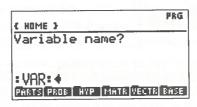
(newline) character once a program has been entered on the stack.)

## **The INPUT Command**

INPUT is used to prompt for data input when the programmer does not want the user to have access to stack operations. Consider the following program:

« "Variable name?" ":VAR:" INPUT »

When this program is executed, the display looks like:



1. The stack area is blanked, and the contents of the string from level 2, Variable name?, are displayed at the top of the stack area. The string from level 2 is called the *prompt string*.

- 2. The contents of the string from level 1, "VAR", are displayed in the command line. The string from level 1 is called the command-line string. Program-entry mode is activated and the insert cursor is positioned after the string. The program is now suspended for data input.
- 3. Program execution is continued by pressing ENTER, which returns the contents of the command line to the stack as a string, called the result string.

The following program, VSPH, calculates the volume of a sphere. VSPH first calculates  $^4/_3\pi$ , then prompts for the radius of the sphere and completes the calculation. Because a partial calculation is already on the stack, VSPH protects the stack by executing INPUT to prompt for the radius. INPUT sets Program-entry mode when program execution pauses for data entry. Subsequent commands are not executed immediately—instead, they are listed in the command line until the user presses ENTER].

Arguments	Results	
1:	1: volume	

Program:	Comments:
«	
4 3 / π * →NUM	Starts the calculation.
"Key in radius"	Builds the prompt string, displayed at the top of the stack area.
H H	Builds the command-line string. In this case, the string is empty, so the command line will be empty.
INPUT	Displays the stack-area prompt, positions the cursor at the start of the command line, and suspends the program for data input (the radius of the sphere).

0BJ÷

Converts the result string into its component object—a real number.

3 ^ \*

Cubes the radius and completes the calculation.

8

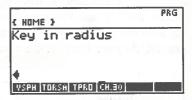
ENTER [ VSPH STO

Stores the program in VSPH.

**Example: Prompting for Data with INPUT.** Execute *VSPH* to calculate the volume of a sphere of radius 2.5 meters.

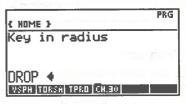
Select the VAR menu and start the program.

VAR WSPH



To show how INPUT protects the stack, press DROP.

(4) DROP



DROP is listed in the command line, but is not executed, so the partial calculation in level 1 is protected.

Press ATTN to restore the command line. Then key in the radius and continue program execution.

ATTN
2.5 ENTER



**Options for the INPUT Command.** In its general form, the level 1 argument for INPUT is a *list* that specifies the content and interpretation of the command line. The list can contain *one or more* of the following parameters, *in any order*:

■ The command-line string, whose contents are placed in the command line for prompting when the program pauses.

- Either a real number, or a list containing two real numbers, that specifies the initial cursor position in the command line:
  - A real number n at the nth character from the left end of the first row (line) of the command line. A positive n specifies the insert cursor; a negative n specifies the replace cursor. 0 specifies the end of the command-line string.
  - A list that specifies the initial row and column position of the cursor: the first number in the list specifies a row in the command line (1 specifies the first row of the command line); the second number counts by characters from the left end of the specified line. 0 specifies the end of the command-line string in the specified row. A positive row number specifies the insert cursor; a negative row number specifies the replace cursor.
- One or more of the parameters ALG,  $\alpha$ , or V, entered as unquoted names:
  - ALG activates Algebraic/Program-entry mode.

  - V verifies if the characters in the result string, without the "delimiters, compose a valid object or objects. If the result-string characters do not compose a valid object or objects, INPUT displays the Invalid Syntax warning and prompts again for data.

The INPUT Default State. You can choose to specify as few as one of the level 1 list parameters. The default states for these parameters are:

- Blank command line.
- Insert cursor placed at the end of the initial command line string.
- Program-entry mode.
- Command-line string not checked for invalid syntax.

If you specify *only* a command-line string for the level 1 argument, you do not need to put it in a list. For example, the previous program, *VSPH*, specifies an empty command-line string for the level 1 argument.

Building the Command-Line String. After the user inputs data to the command line and presses [ENTER] to resume program execution, the contents of the command line are returned to level 1 as the result string. To process the input, the program may at some point execute OBJ— to convert the result string to a valid object or objects. The program can accomplish this by specifying a command-line string of known form and then taking appropriate action after the result string is returned to level 1:

- The program can specify an *empty* command-line string. In this case, the result string consists *only* of the input. The program *VSPH* on page 525 uses this method.
- The program can specify a command-line string whose characters form the tag and delimiters for a tagged object. (See page 87 for a discussion of tagged objects.) In this case, the input completes the tagged object. The program TINPUT on page 529 uses this method.
- The program can specify a command-line string whose characters form a message. In this case, the program subtracts those characters from the result string to leave only the input in the string in string form. The program SSEC on page 531 uses this method.

In the first two cases, the V parameter can also be specified as part of the level 1 argument to specify that INPUT reprompt for data if the contents of the result string are not valid objects.

The following program, TINPUT, executes INPUT to prompt for the inner and outer radii of a torus, then calls TORSA (chapter 25, page 481) to calculate its surface area. TINPUT prompts for a and b in a two-row command line; the level 1 argument for INPUT is list that contains:

- The command-line string.
- An imbedded list specifying the initial cursor position.
- The V parameter to check for invalid syntax in the result string.

The command-line string forms the tags and delimiters for two tagged objects. The list does not specify the entry mode, so Program-entry mode is selected by default.

Arguments	Results	
1:	1: area	

Program:	Comments:
«	
"Key in a, b"	Builds the level 2 string, displayed at the top of the stack area.
<	Starts the level 1 list argument.
":a:■:b:" (1 0) V	The level 1 list contains a command- line string, a list, and the verify- syntax specification. (To key in the string, press b. Lie a limit in the string, press lie in lie in the string will be shown on the stack, the string will be shown on one line, with the character indicating the newline character.)  The imbedded list positions the insert cursor in row 1 just after as V specifies to check for invalid syntax in the result string.
3	Ends the level 1 list argument.
INPUT	Displays the stack string and command-line string, positions the cursor as specified by the list in the level 1 argument, and, by default, sets Program-entry mode. Then suspends program execution for data. Checks the resultant string for syntax errors.
OBJ÷	Converts the string into its component objects (in this case, two tagged objects).

Calls TORSA to calculate the surface area.

TORSA

\*

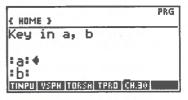
[ENTER] [] TINPUT [STO]

Stores the program in TINPUT.

**Example: Prompting for Data with Input.** Execute TINPUT to calculate the surface area of a torus of inner radius a = 10 cm and outer radius b = 20 cm.

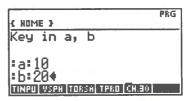
Select the VAR menu and start the program.

#### VAR TINPU



Key in the value for a and press  $\P$  to move the cursor to the next prompt in the command line. Then key in the value for b.

10 🔻 20



Continue program execution.

ENTER

1: 2960.88132033 TINPU VSPH TORSA TERU CH.SO

The following program executes INPUT to prompt for a social security number, then extracts in string form the first three digits and last four digits from the result string. The level-1 argument for INPUT specifies:

- A command-line string.
- The replace cursor positioned at the start of the prompt string (-1).
   The replace cursor lets the user "fill in" the command line string, using ▶ to skip over the dashes in the social-security number.
- By default, Program-entry mode.
- By default, no verification of object syntax—the dashes in the socialsecurity number are not valid characters outside the string delimiters.

Arguments	Results	
2:	2: " first three digits"	
1:	1: "last four digits"	

Program:	Comments:
«	
"Key in S.S. #"	Builds the level 2 string, displayed at the top of the stack area.
(" " -1 )	Builds the level 1 argument for INPUT. (Key in 3 spaces between the first "delimiter and the first -, two spaces between the two -'s, and 4 spaces between the last - and the ending "delimiter.)
INPUT	Suspends the program for data.
DUP 1 3 SUB SWAP 8 11 SUB	Copies the result string, then extracts the first three and last four digits in string form.
*	
ENTER 'SSEC STO	Stores the program in SSEC.

# **Labeling Program Output**

A descriptive tag or message can make program output more recognizable.

## **Using Tagged Objects as Data Output**

You can label a program result using the →TAG command. →TAG (PRG OBJ →TAG) takes two arguments: any object from level 2, and a name, string, or real number (the tag) from level 1.

The following program TTAG is identical to TINPUT, except that it tags the result.

Program:

**Comments:** 

**%**.

"Key in a, b"

{ ":a:::b:" {1 0} V }

INPUT OBJ→

TORSA

'AREA' Builds the tag, in this case a name.

→ TAG Joins the tag to the object in level 2, the program result, to create the

tagged object.

\*

ENTER TTAG STO

Stores the program in TTAG.

**Example: Using a Tagged Object for Data Output.** Execute TTAG to calculate the area of a torus of inner radius a = 1.5 and b = 1.85.

Select the VAR menu and start the program. Supply the values for a and b and continue program execution. The answer is returned as a tagged object to the stack.

VAR TTAG 1.5 ▼ 1.85 ENTER

1: AREA: 11.5721111603

## **Using String Commands to Label Data Output**

You can use string commands and DISP to label and display an object that has been returned to level 1 of the stack:

- 1. Convert the object to a string with →STR (PRG □BU □ STR).
- 2. Enter a labeling string on the stack.
- 3. Swap the two strings on the stack, then concatenate them (SWAP +).
- 4. Display the resultant string (n DISP).

The following program TSTRING is identical to TINPUT, except that it converts the program result to a string and appends a labeling string to it.

#### Program:

#### Comments:

«	
"Key in a, b"	
<pre>{ ":a:::b:" (1 0) V }</pre>	
INPUT OBJ>	
TORSA	
⇒STR	Converts the result to a string.
"Area = "	Enters the labeling string.
SWAP +	Swaps the positions of the two strings on the stack and adds them.
CLLCD 1 DISP 1 FREEZE	Displays the resultant string, without its delimiters, in line 1 of the display.
»	
ENTER TSTRING STO	Stores the program in TSTRING.

**Example: Labeling Data Output.** Execute TSTRING to calculate the area of the torus in the previous example (a = 1.5, b = 1.85).

Select the VAR menu and start the program. Supply the values for a and b and continue program execution. The labeled answer is displayed in the status area.

```
VAR TSTRI
1.5 ▼ 1.85
ENTER
```

```
Area = 11.5721111603
4:
3:
2:
1:
1:
```

## **Pausing to Display Data Output**

The WAIT command (PRG CTRL NXT WAIT) suspends program execution for x seconds, where x is a positive real number from level 1. You can use WAIT with DISP to display messages during program execution—for example, to display intermediate program results.

WAIT interprets arguments 0 and -1 differently—see "Commands That Return a Key Location" on page 539.

# **Using Menus in Programs**

Applications menus like the SOLVE and PLOT menus, as well as the VAR and CST menus, can be activated and used in a program as they are during normal keyboard operations.

## Displaying a Built-In Menu

To display a built-in menu in a program, execute the MENU command (PRG CTRL NXT MENU) with the numeric argument that corresponds to that built-in menu. The table in Appendix D lists all the HP 48 menus and their corresponding menu numbers. For example, 20 MENU activates page 1 of the MODES menu. You can specify a particular page of a menu by supplying the argument in the form xx.yy, where xx is the menu number, and yy is the page number.

The following program activates the third page of the MODES menu and asks you to set the angle mode.

« 20.03 MENU "Select Angle Mode" PROMPT »

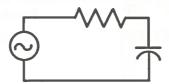
RCLMENU (MODES NXT ROLM) returns the menu number of the currently displayed menu.

## **Custom Menus in Programs**

In chapter 15 you learned how to build a custom menu by supplying a list argument for MENU. In programs, you can construct custom menus to:

- Emulate built-in applications like the HP Solve application.
- Prompt you to make decisions.

**Emulating Built-In Applications.** The following program, *EIZ*, constructs a custom menu to emulate the HP Solve application for capacitive electrical circuits.



Application of Ohm's law to this circuit results in the following expression:

$$E = IZ$$

where

E is the circuit voltage.I is the circuit current.Z is the circuit impedance.

Because the voltage, current, and impedance are complex numbers, you cannot use the HP Solve application to find solutions. The custom menu in *EIZ* assigns a *direct* solution to the left-shifted menu key for each variable, and assigns *store* and *recall* functionality to the unshifted and right-shifted keys—the key actions are analogous to the HP Solve application.

#### \* Sets Degrees mode. Sets flags -15 DEG and -16 to display complex numbers -15 SF -16 SF 2 FIX in polar form. Sets the display mode to 2 Fix. { Starts the list for the custom menu. Builds menu key 1, labeled E. { "E" { « 'E' STO » When you press E, the object « I Z \* DUP 'E' STO in level 1 is stored in variable E. "E" →TAG When you press , the CLLCD 1 DISP product of I and Z is calculated, 1 FREEZE >> stored in variable E, and displayed as «E»)) a tagged object. When you press $\vdash$ E, the object stored in E is returned to level 1. { "I" { « 'I' STO » Builds menu key 2. « E Z / DUP 'I' STO "I" →TAG CLLCD 1 DISP 1 FREEZE >> « I » ) ) { "Z" { « 'Z' STO » Builds menu key 3. « E I / DUP 'Z' STO "Z" →TAG CLLCD 1 DISP 1 FREEZE >> « Z » ) ) Ends the list. 3 Displays the custom menu. MENU » ENTER ( EIZ STO) Stores the program in EIZ.

Comments:

Program:

**Example: Emulating a Built-In Application.** A 10-volt power supply at phase angle 0° drives an RC circuit. A current of .37 A at phase angle 68° is measured. What is the impedance of the circuit?

Select the VAR menu and start EIZ.

VAR EIZ

Key in the value for the voltage.

**4** ( ) 10 ( ) ( ) ( ) ( ) ( )

10404

Store the value for the voltage. Then key in and store the value for the current. Solve for the impedance.

E (年)(1) .37 (尹本) 68 I (年) Z :Z: (27.03,4-68.00) 4: 3: 2: 1:

If the current amplitude is doubled and the impedance remains constant, what is the complex voltage?

**♣**().74 **▶** △ 68

:E: (20.00, 4-1.36E-10)
4:
3:
2:
1:

Recall the value of Z to the stack.

P Z

1: (27.03, <-68.00)

**Prompting for a Choice.** A custom menu can prompt the user to make a decision during program execution.

The program WGT in this section calculates the weight of an object in either English or SI units. WGT builds a custom menu that prompts the user to select the desired unit system. Here is the defining list for the custom menu:

```
{
    ( "ENGL" « "ENTER Mass
    in LB" PROMPT
    32.2 * » )
    ( "SI" « "ENTER Mass
    in KG" PROMPT
    9.81 * » )
}
```

If you store this list in variable LST, program WGT is simply:

#### Program:

#### **Comments:**

≪

LST MENU

Displays the custom menu stored in *LIST*.

>

ENTER | WGT STO

Stores the program in WGT.

The custom menu defined by WGT remains active until you select a new menu, so you can do as many calculations as you want.

Note that the custom menu defined by WGT (and the custom menu defined by EIZ) is automatically stored in variable CST, replacing the previous custom menu—when you press CST after the program ends, the menu defined by WGT is displayed.

**Example: Using a Custom Menu to Make a Choice.** Use WGT to calculate the weight of an object of mass 12.5 kg.

Select the VAR menu and start the program.

VAR WGT

ENGL SI

Select the SI unit system.

SI

ENTER Mass in KG 4: 3: 2: 1: engl si Key in the mass and continue program execution.



## **Building a Temporary Menu**

The TMENU command (MODES NXT TMEN) works just like MENU, except that list arguments do not replace the contents of CST and so leave the current custom menu unchanged. Note that the temporary menu remains active until a new menu is selected, even after the program ends. To programmatically restore the previous menu, execute @ MENU.

The program « LIST TMENU » is similar to WGT, except that it builds a temporary menu to prompt for the unit-system choice.

# **Commands That Return a Key Location**

## The WAIT Command with Argument 0

If you supply 0 as the argument for WAIT, the command suspends program execution until a valid keystroke is executed. It then returns the three-digit location number that defines where the key is on the keyboard and restarts program execution. (See section "Making User-Key Assignments" on page 217 in chapter 15.)

(Note that  $\leftarrow$ ,  $\curvearrowright$ ,  $\curvearrowright$ ,  $\curvearrowright$ , or  $\curvearrowright$  do not by themselves constitute a valid keystroke.)

## The WAIT Command with Argument -1

The WAIT command with argument -1 works just like it does with argument 0, except that the currently specified menu is also displayed. This lets you build and display a prompting menu while the program is paused. (Note that a menu built with MENU or TMENU is not normally displayed until the program ends or is halted with HALT.)

#### The KEY Command

A program can prompt for a simple "yes-no" decision using the KEY command in an indefinite loop, and a comparison test. (Indefinite looping structures are covered in chapter 27. Tests are covered in chapter 26.) When the loop begins, KEY simply returns a false result (2) to level 1 until a key is pressed. Once a key is pressed, KEY returns the two-digit location number that defines where the key is on the keyboard and returns a true result (1) to level 1. For example, when you use KEY in an indefinite loop and press ENTER, KEY returns 51 to level 2 and true result 1 to level 1.

The following program segment returns 1 to level 1 if + is pressed, or 0 to level 1 if any other key is pressed:

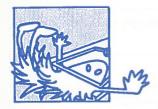
« ... DO UNTIL KEY END 95 SAME ... »

(Note that KEY returns only a two-digit location number RowColumn, unlike WAIT, which returns a three-digit location number that identifies shifted and alpha keys. Thus, if you press the key, KEY returns 71, while WAIT does not interpret itself as a valid keystroke.)

# **Turning the HP 48 Off from a Program**

The OFF command turns the HP 48 off. If executed from a program, the program will resume when the calculator is turned back on.

# **Error Trapping**



When you attempt an invalid operation from the keyboard, the operation is not executed and an error message is displayed. For example, if you execute + with a vector and a real number on the stack, the HP 48 returns the message:

+ Error: Bad Argument Type

and, assuming that Last Arguments is enabled, returns the arguments to the stack. In a program, the same thing happens, but program execution is also aborted. Consider the following program:

« "KEY IN a AND b" "" INPUT OBJ→ + »

If you execute this program and supply a vector and a real number at the prompt, the program displays the Bad Argument Type error message and aborts execution at the + command. To supply new arguments, you must restart the program. For a short program like the one above, this method of error recovery presents little problem. However, when executing a program that performs time consuming calculations, or that has numerous stops for intermediate data entry, it may be inconvenient to restart the program at the beginning each time an error occurs.

You can enable a program to *continue* execution after an error has occurred by building an *error trap*. You can construct an error trap with one of the following conditional structures:

- IFERR...THEN...END.
- IFERR...THEN...ELSE...END.

The IFERR command is located on page 3 of the PRG BRCH menu.

The following commands enhance error-trap structures:

#### **Error Trapping Commands**

Keys	Programmable Command	Description
PRG CT	RL (page 3):	KEY EM OF SHEET WELL
DOERR	DOERR	Executes a user-specified error. The calculator behaves just as if an ordinary error has occurred — if the error is not trapped in an IFFER structure, DOERR displays a message and abandons program execution.
<b>= 2.73</b> 1	ERRN	Returns the error number, as a binary integer, of the most recent error.  Returns #0 if the error number was cleared by ERR0.
=R44M	ERRM	Returns the error message (a string) for the most recent error. Returns empty string if the error number was cleared by ERR0.
ERR0	ERR0	Clears the last error number, so that a subsequent execution of ERRN returns #0. Also clears the last error message.

## The IFERR...THEN...END Structure

The syntax of IFERR...THEN...END is

IFERR trap-clause THEN error-clause END

If an error occurs during execution of the trap-clause, the error is ignored, the remainder of the trap-clause is discarded and program execution jumps to the error-clause. The commands in the error-clause are executed only if an error is generated during execution of the trap-clause.

As a typing aid, press TIFERR to key in:

IFERR THEN END

**Example: An IFERR...THEN...END Structure.** Recall the following program from chapter 27, page 512.

« WHILE DUP TYPE 3 == REPEAT X+ END »

The program takes any number of vectors or arrays from the stack and adds them to the statistics matrix. However, the program errors if a vector or array with a different number of columns is encountered. In addition, if only vectors or arrays with the same number of columns are on the stack, the program errors after the last vector or array has been removed from the stack.

In the following version, the program simply attempts to add the level 1 object to the statistics matrix until an error occurs. At that point, it "gracefully" ends by displaying the message DONE.

Program:	Comments:
«	
IFERR	Starts the trap-clause.
WHILE	Starts the test-clause of the nested loop.
1	1 is a true result, so executes the loop-clause until an error occurs.
REPEAT	Starts the loop clause.
Σ+	Adds the vector or array to the statistics matrix.
END	Ends the nested loop.
THEN	If an error does occur on execution of $\Sigma$ +
"DONE" 1 DISP 1 FREEZE	displays the message DONE in the status area.
END	Ends the error trap.

# The IFERR...THEN...ELSE...END Structure

The syntax of IFERR...THEN...ELSE...END is:

IFERR trap-clause THEN error-clause ELSE normal-clause END

If an error occurs during execution of the trap-clause, the error is ignored, the remainder of the trap-clause is discarded and program execution jumps to the error-clause. If no error occurs, execution jumps to the normal-clause at the completion of the trap-clause.

### As a typing aid, press FERR to key in:

IFERR THEN ELSE END

**Example:** An IFERR...THEN...ELSE...END Structure. The following program prompts for two numbers, then adds them. If only one number is supplied, the program displays an error message and prompts again.

Program:	Comments:
« and a second of the contract of	
DO	Begins the outer loop.
"KEY IN a AND b" " " INPUT OBJ>	Prompts for two numbers.
UNTIL	Starts the test clause
IFERR	Starts the error trap.
+ 300 0 4 100 000 000 000	Adds the contents of levels 1 and 2.
THEN	If an error occurs
ERRM 5 DISP 2 WAIT 0	executes ERRM to display the Too Few Arguments error message for two seconds, then leaves $\emptyset$ (false) on the stack for the outer-loop END.
ELSE	If an error does not occur
1	leaves 1 (true) on the stack for the outer-loop END.
END	Ends the error trap.

END

Ends the outer loop. If the error trap left @ on the stack, this END returns program execution to the prompt for numbers. Otherwise, the program ends.

\*

## **User-Defined Errors**

You may want to generate an error in a program when an error would not normally occur. For example, you might want an error to occur if the sum of the two numbers on the stack is greater than 10. You can do this with the DOERR command. DOERR causes a program to behave exactly as if a normal error has occurred during execution. The DOERR error can be trapped in an IFERR structure; if it is not, program execution is abandoned at the location of the DOERR command. DOERR takes one argument from the stack, either:

- A string, in which case the string is used as the message. (ERRM returns this string, and ERRN returns #70000h.)
- A real number or binary integer, in which case the corresponding built-in error message is displayed. (ERRM and ERRN return the corresponding error message and number, respectively.) DOERR is equivalent to [ATTN]; that is, program execution is aborted and no message is displayed. (In this case, the values returned by ERRM and ERRN are unchanged from their previous values.)

The following program aborts execution if there are three objects in the level 1 list.

```
«

OBJ+

IF 3 SAME

THEN "3 OBJECTS IN LIST" DOERR

END

*
```

In this program, DOERR abandons program execution. Alternatively, you can execute DOERR in the trap-clause of an error trap to enable program execution to continue.

# **More Programming Examples**



The programs in this chapter demonstrate programming concepts introduced in the previous chapters. Some new concepts are also introduced. The programs are intended to both improve your programming skills and provide supplementary functions for your calculator.

At the end of each program, the checksum and the program size in bytes are listed. The checksum is a binary integer that uniquely identifies the program based on its contents. To verify that you've keyed the program in correctly, execute the BYTES command (MEMORY) BYTES) with the program name in level 1. The checksum for the program is returned to level 2, and its size in bytes is returned to level 1. (If you execute BYTES with the program object in level 1, before storing the program in its name, you'll get a different byte count returned to level 1.)

### **Fibonacci Numbers**

This section includes three programs — two demonstrate an approach to the following problem:

Given an integer n, calculate the nth Fibonacci number  $F_n$ , where:

$$F_0 = 0$$
,  $F_1 = 1$ ,  $F_n = F_{n-1} + F_{n-2}$ 

- FIB1 is a user-defined function that is defined recursively—its defining procedure contains its own name. FIB1 is short.
- FIB2 is a user-defined function with a definite loop. It's longer and more complicated than FIB1, but it's faster.

The third program, FIBT, calls both FIB1 and FIB2, and calculates the execution time of each subprogram.

## FIB1 (Fibonacci Numbers, Recursive Version)

Arguments	Results
1: n	1: F <sub>n</sub>

#### Techniques.

- IFTE (If-Then-Else function). The defining procedure for FIB1 contains the conditional function IFTE, which can take its argument either from the stack or in algebraic syntax. (FIB2 uses the conditional structure IF... THEN... ELSE... END.)
- Recursion. The defining procedure for *FIB1* is written in terms of *FIB1*, just as  $F_n$  is defined in terms of  $F_{n-1}$  and  $F_{n-2}$ .

#### Program:

#### **Comments:**

«

→ n

Defines local variable n.

1

Begins the defining procedure, an algebraic expression.

IFTE(n≤1,

If  $n \leq 1 \dots$ 

p.500

n,

Emitra Ba

... then  $F_n = n$  ...

FIB1(n-1)+FIB1(n-2))

... else  $F_n = F_{n-1} + F_{n-2}$ .

Ends the defining procedure.

\*

ENTER ( FIB1 STO

Enters the program, then stores it in FIB1.

Checksum:

# 41467d

**Bytes:** 

113.5

**Example.** Calculate  $F_6$ . Calculate  $F_{10}$  using algebraic syntax.

First calculate F<sub>6</sub>.

VAR

6 FIB1

1:

FIBT | PAD

Next calculate F<sub>10</sub> using algebraic syntax.

() FIB1 (-) 10 EVAL

1:

50 PAO PRESE

## FIB2 (Fibonacci Numbers, Loop Version)

Arguments	Results
1: n	1: F <sub>n</sub>

#### Techniques.

- IF...THEN...ELSE...END. FIB2 uses the program-structure form of the conditional. (FIB1 uses IFTE.)
- START...NEXT (definite loop). To calculate  $F_n$ , FIB2 starts with  $F_0$  and  $F_1$  and repeats a loop to calculate successive  $F_i$ 's.

Program:	Comments:
«	
→ n	Creates a local variable.
«	Begins the defining procedure, a program.
IF n 1 ≤	If $n \le 1 \dots$
THEN n	then $F_n = n$ .
ELSE	Begins the ELSE clause.
0 1	Puts F <sub>0</sub> and F <sub>1</sub> on the stack.
2 n	From 2 to <i>n</i>
START	does the following loop:
DUP	Makes a copy of the latest $F$ (initially $F_1$ ).
ROT	Moves the previous F (initially $F_0$ ) to level 1.
+	Calculates the next F (initially F <sub>2</sub> ).

NEXT

Repeats the loop.

SWAP DROP

Drops  $F_{n-1}$ .

END

Ends the ELSE clause.

50.

Ends the defining procedure.

•••

Ends the defining procedure

Ends the program.

ENTER | FIB2 STO

Enters the program, then stores it in FIB2.

Checksum:

# 51820d

Bytes:

89

**Example.** Calculate  $F_6$  and  $F_{10}$ . Note that *FIB2* is faster than *FIB1*.

Calculate F<sub>6</sub>.

**VAR** 

6 F 32

1: BÖISP FIB1 FIB2 FIBT PHO PRESE

Calculate F<sub>10</sub>.

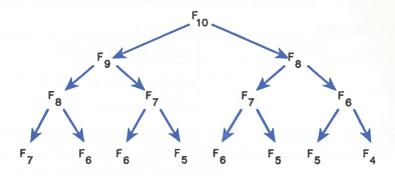
10 F #32

2: 8 1: 55 [BOJSP] FIB1 | FIB2 | FIB1 | PAG | PRESE

## FIBT (Comparing Program-Execution Time)

FIB1 calculates intermediate values  $F_i$  more than once, while FIB2 calculates each intermediate  $F_i$  only once. Consequently, FIB2 is faster. The difference in speed increases with the size of n because the time required for FIB1 grows exponentially with n, while the time required for FIB2 grows only linearly with n.

The diagram below shows the beginning steps of FIB1 calculating  $F_{10}$ . Note the number of intermediate calculations: 1 in the first row, 2 in the second row, 4 in the third row, and 8 in the fourth row.



FIBT executes the TICKS command to record the execution time of FIB1 and FIB2 for a given value of n.

Arguments	Results
1: n	3: F <sub>n</sub> 2: FIB1 execution time: z 1: FIB2 execution time: z

#### Techniques.

Program:

- Structured programming. FIBT calls both FIB1 and FIB2.
- Programmatic use of calculator clock. FIBT executes the TICKS command to record the start and finish of each subprogram.
- Interactive programming. FIBT tags each execution time with a descriptive message.

«	
DUP TICKS SWAP FIB1 SWAP TICKS SWAP	Copies $n$ , then executes $FIB1$ , recording the start and stop time.
- B÷R 8192 ∕	Calculates the elapsed time, converts it to a real number, and converts that number to seconds. Leaves the

Comments:

answer returned by FIB1 in level 2.

"FIB1 TIME" ⇒TAG

Tags the execution time.

ROT TICKS SWAP FIB2

Executes FIB2, recording the start

SWAP DROP SWAP

and stop time.

- B+R 8192 /

Drops the answer returned by FIB2 (FIB1 returned the same answer). Calculates the elapsed time for FIB2

and converts to seconds.

"FIB2 TIME"

Tags the execution time.

**→TAG** 

TICKS

ENTER () FIBT (STO)

Stores the program in FIBT.

Checksum:

# 22248d

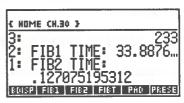
Bytes:

135

**Example.** Calculate F<sub>13</sub> and compare the execution time for the two methods.

Select the VAR menu and do the calculation.

VAR 13 FIBT.



 $F_{13}$  is 233. FIB2 takes 0.13 seconds to execute. FIB1 takes 33.9 seconds. (Your results may differ depending on the contents of memory in your calculator.)

# Displaying a Binary Integer

This section contains three programs:

- PAD is a utility program that converts an object to a string for right-justified display.
- PRESERVE is a utility program for use in programs that change the calculator's status (angle mode, binary base, and so on).
- BDISP displays a binary integer in HEX, DEC, OCT, and BIN bases. It calls PAD to show the displayed numbers right-justified, and it calls PRESERVE to preserve the binary base.

## **PAD (Pad with Leading Spaces)**

PAD converts an object to a string and, if the string contains fewer than 23 characters, adds spaces to the beginning.

When a short string is displayed with DISP, it appears *left-justified*; its first character appears at the left end of the display. The position of the last character is determined by the length of the string. By adding spaces to the beginning of a short string, *PAD* moves the position of the last character to the right. When the string (including leading spaces) is 23 characters long, it appears *right-justified*; its last character appears at the right end of the display. *PAD* has no effect on strings that are longer than 22 characters.

Arguments	Results
1: object	1: "object"

#### Techniques.

- WHILE ... REPEAT ... END (indefinite loop). The WHILE clause contains a test that determines whether to execute the REPEAT clause and test again (if true) or to skip the REPEAT clause and exit (if false).
- String operations. *PAD* demonstrates how to convert an object to string form, count the number of characters, and concatenate two strings.

Program:	Comments:
«	
⇒STR	Makes sure the object is in string form. (Strings are unaffected by this command.)
WHILE	Begins WHILE clause.
DUP SIZE 22 <	Does the string contain fewer than 23 characters?
REPEAT	Begins REPEAT clause.
" " SWAP +	Adds a leading space.
END	Ends REPEAT clause.

ENTER [] PAD STO	Enters the program, then stores it in
	PAD.

Checksum: # 38912d Bytes: 61.5

ENTER PAR STO

PAD is demonstrated in the program BDISP.

# **PRESERVE (Save and Restore Previous Status)**

Given a program on the stack, PRESERVE stores the current calculator (flag) status, executes the program, and then restores the previous status.

Arguments	Results
1: « program »	1: (result of program)
1: 'program name'	1: (result of program)

#### Techniques.

- RCLF and STOF. PRESERVE uses RCLF (recall flags) to record the current status of the calculator in a binary integer and STOF (store flags) to restore the status from that binary integer.
- Local-variable structure. PRESERVE creates a local variable structure to remove the binary integer from the stack briefly; its defining procedure simply evaluates the program argument, then puts the binary integer back on the stack and executes STOF.

Program:	Comments:
«	
RCLF	Recalls the list of two 64-bit binary integers representing the status of the 64 system flags and 64 user flags.
→ f	Stores the list in local variable $f$ .
«	Begins the defining procedure.
EVAL	Executes the program placed on the stack as the level 1 argument.
f STOF	Puts the list back on the stack, then restores the status of all flags.
*	Ends the defining procedure.
<b>»</b>	
ENTER [] PRESERVE STO	Enters the program, then stores it in <i>PRESERVE</i> .
01-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-	

**Checksum:** # 21528d **Bytes:** 46.5

PRESERVE is demonstrated in the program BDISP.

## **BDISP (Binary Display)**

BDISP displays a (real or binary) number in HEX, DEC, OCT, and BIN bases.

	Arguments	Results
1:	# n	1: # n
1:	n	1: n

#### Techniques.

- IFERR...THEN...END (error trap). To accommodate realnumber arguments, BDISP includes the command R→B (real-tobinary). However, this command causes an error if the argument is already a binary integer. To maintain execution if an error occurs, the R→B command is placed inside an IFERR clause. No action is required when an error occurs (since a binary number is an acceptable argument), so the THEN clause contains no commands.
- Enabling LASTARG. In case an error occurs, LASTARG must be enabled to return the argument (the binary number) to the stack.

  BDISP clears flag -55 to enable the LASTARG recovery feature.
- FOR... NEXT loop (definite loop with counter). BDISP executes a loop from 1 to 4, each time displaying n (the number) in a different base on a different line. The loop counter (named j in this program) is a local variable. It is created by the FOR... NEXT program structure (rather than by a → command) and it is automatically incremented by NEXT.
- Unnamed programs as arguments. A program defined only by its « and » delimiters (not stored in a variable) is not automatically evaluated; it is simply placed on the stack and may be used as an argument for a subroutine. BDISP demonstrates two uses for unnamed program arguments.
  - 1. BDISP contains a main program argument and a call to PRESERVE. This program argument goes on the stack and is executed by PRESERVE.

2. There are four program arguments that "customize" the action of the loop. Each program argument contains a command to change the binary base, and each iteration of the loop evaluates one of these arguments.

When BDISP creates a local variable for n, the defining procedure is an unnamed program. However, since this program is a defining procedure for a local variable structure, it is automatically executed.

#### Required Programs.

- PAD (page 555) expands a string to 23 characters so that DISP shows it right-justified.
- PRESERVE (page 556) stores the current status, executes the main nested program and restores the status.

Program:	Comments:
«	
«	Begins the main nested program.
DUP	Makes a copy of $n$ .
-55 CF	Clears flag -55 to enable LASTARG.
IFERR	Begins error trap.
R→B	Converts $n$ to a binary integer.
THEN	If an error occurred
END	do nothing (there are no commands in the THEN clause).
→ n	Creates a local variable n.
«	Begins the defining program for the local variable structure.
CLLCD	Clears the display.
« BIN »	Writes the nested program for BIN.

Writes the nested program for OCT.
DEC > Writes the nested program for DEC.
HEX > Writes the nested program for HEX.
Sets the first and last counter values.
FOR j
Starts the loop with counter j.
Evenues one of the pested base.

EVAL Executes one of the nested base programs (initially the one for HEX).

n  $\rightarrow$ STR Makes a string showing n in the current base.

PAD Pads the string to 23 characters.

j DISP Displays the string in the jth line.

NEXT Increments j and repeats the loop.

Ends the defining procedure.

3 FREEZE Freezes the status and stack areas.

Ends the main nested program.

PRESERVE Stores the current status, executes the main nested program, and

restores the status.

ENTER [] BDISP STO Enters the program, then stores it in BDISP.

**Checksum:** # 18055d **Bytes:** 191

**Example.** Switch to DEC base, display # 100 in all bases, and check that BDISP restored the base to DEC.

Clear the stack and select the MTH BASE menu. Make sure the current base is DEC and enter # 100.

CLR

MTH BASE

DEC

(→) # 100 ENTER

Execute BDISP.

VAR BDISP

1: # 109d HEX DEC - OCT SIN STUS ROUS

# 64h # 100d # 144o # 1100100b

Return to the normal stack display and check the current base.

ATTN

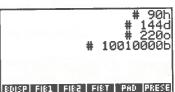
HEX DEC - DOT BIN STUS ROWS

MTH BASE

Although the main nested program left the calculator in BIN base, *PRESERVE* restored DEC base.

To check that BDISP also works for real numbers, try 144.

VAR 144 BDISP



## **Median of Statistics Data**

This section contains three programs:

- SORT orders the elements of a list.
- LMED calculates the median of a sorted list.
- MEDIAN uses SORT and LMED to calculate the median of the current statistics data.

## **SORT (Sort a List)**

SORT sorts a list of real numbers into ascending order.

Arguments	Results
1: { list }	1: { sorted list }

#### Techniques.

- Bubble sort. Starting with the first and second numbers in the list, SORT compares adjacent numbers and moves the larger number toward the end of the list. This process is done once to move the largest number to the last position in list, then again to move the next largest to the next-to-last position, and so on.
- Nested definite loops. The outer loop controls the stopping position each time the process is done; the inner loop runs from 1 to the stopping position each time the process is done.
- Nested local-variable structures. SORT contains two local-variable structures, the second inside the defining procedure (a program) of the first. This nesting is done for convenience; it's easier to create the first local variable as soon as its value is computed, thereby removing its value from the stack, rather than computing both values and creating both local variables at once.
- FOR... STEP and FOR... NEXT (definite loops). SORT uses two counters: -1 STEP decrements the counter for the outer loop each iteration; NEXT increments the counter for the inner loop by 1 each iteration.

#### Comments: Program: From the next-to-last position to the DUP SIZE 1 - 1 first position ... ... begins the outer loop with FOR .i counter j. From the first position to the jth 1 j position ... ... begins the inner loop with FOR k counter k. Gets the kth number in the list and k GETI → n1 stores it in a local variable $n_1$ . Begins the defining procedure (a « program) for the outer local variable structure. Gets the next number in the list and GETI → n2 stores it in a local variable $n_2$ . Begins the defining procedure (a program) for the inner local variable structure. Drops the index returned by GETI. DROP If the two numbers are in the wrong IF n1 n2 > order ... ... then does the following: THEN ... puts the second one back in the k n2 PUTI kth position; ... puts the kth one back in the next nt PHT position. Ends THEN clause. END

» Ends inner defining procedure.

Ends outer defining procedure.

NEXT Increments k and repeats the inner

loop.

-1 STEP Decrements j and repeats the outer

loop.

>>

ENTER [] SORT STO Enters the program, then stores it in

SORT.

Checksum: # 15011d

Bytes: 144

**Example.** Sort the list { 83125 }.

Select the VAR menu, key in the list, and execute SORT.

VAR

(→) (→) 8 SPC 3 SPC

1 SPC 2 SPC 5 ENTER

SORT

1: { 1 2 3 5 8 } SORT LMED MEDINIOURN HULTI EXCO

## LMED (Median of a List)

Given a sorted list, *LMED* returns the median. If the list contains an odd number of elements, the median is the value of the center element. If the list contains an even number of elements, the median is the average value of the elements just above and below the center.

Arguments	Results
1: { sorted list }	1: median of sorted list

## Techniques.

■ FLOOR and CEIL. For an integer, FLOOR and CEIL both return that integer; for a noninteger, FLOOR and CEIL return successive integers that bracket the non-integer.

Program:	Comments:
«	
DUP SIZE	Copies the list, then finds its size.
1 + 2 /	Calculates the center position in the list (fractional for even-sized lists).
→ p	Stores the center position in local variable $p$ .
«	Begins the defining procedure (a program) for the local variable structure.
DUP	Makes a copy of the list.
P FLOOR GET	Gets the number at or below the center position.
SWAP	Moves the list to level 1.
P CEIL GET	Gets the number at or above the center position.
+ 2 /	Calculates the average of the two numbers at or near the center position.
<b>»</b>	Ends the defining procedure.
*	
ENTER [] LMED STO	Enters the program, then stores it in <i>LMED</i> .

**Checksum:** # 3682d **Bytes:** 77

**Example.** Calculate the median of the list you sorted using SORT.

Put the list on the stack if necessary, select the VAR menu, and execute *LMED*.

(4) (3) 12358 ENTER 1: SORT LIMED MEDIA COMMINIULTH EXCO

## **MEDIAN (Median of Statistics Data)**

MEDIAN returns a vector representing the medians of the columns of the statistics data.

Arguments	Results
1:	1: [X <sub>1</sub> X <sub>2</sub> X <sub>m</sub> ]

#### Techniques.

Arrays, lists, and stack elements. MEDIAN extracts a column of data from ΣDAT in vector form. To convert the vector to a list, MEDIAN puts the vector elements on the stack and then combines them into a list. From this list the median is calculated using SORT and LMED.

The median for the *m*th column is calculated first, and the median for the first column is calculated last, so as each median is calculated, it is moved to the stack level above the previously calculated medians.

After all medians are calculated and positioned correctly on the stack, they're combined into a vector.

■ FOR... NEXT (definite loop with counter). MEDIAN uses a loop to calculate the median of each column. Because the medians are calculated in reverse order (last column first), the counter is used to reverse the order of the medians.

#### Required Programs.

- SORT (page 562) arranges a list in ascending order.
- LMED (page 564) calculates the median of a sorted list.

#### Comments: Program: « Puts a copy of the current statistics RCLZ matrix $\Sigma DAT$ on the stack for safekeeping. Puts the list $\{n m\}$ on the stack, DUP SIZE where n is the number of rows in $\Sigma DAT$ and m is the number of columns. Puts n and m on the stack. Drops the OBJ→ DROP list size. Creates local variables for n and m. → n m Begins the defining procedure (a program) for the local variable structure. Transposes $\Sigma DAT$ . Now n is the 'ΣDAT' TRN number of columns in $\Sigma DAT$ and m is the number of rows. (To key in the $\Sigma$ character, press $\triangleright \Sigma$ , then delete the parentheses.) Specifies the first and last rows. For each row, does the following: FOR j Extracts the last row in $\Sigma DAT$ . Σ-Initially this is the mth row, which corresponds to the mth column in the original $\Sigma DAT$ . (To key in the $\Sigma$ command, press (STAT) **(5)**

OBJ→ DROP	Puts the row elements on the stack.
	Drone the index list [ n ] since n is

Drops the index list  $\{n\}$ , since n is already stored in a local variable.

n →LIST Makes an n-element list.

SORT Sorts the list.

LMED Calculates the median of the list.

j ROLLD Moves the median to the proper

stack level.

NEXT Increments j and repeats the loop.

m → ARRY Combines all the medians into an

*m*-element vector.

Ends the defining procedure.

SWAP Moves the original  $\Sigma DAT$  to level 1.

STO  $\Sigma$  Restores  $\Sigma DAT$  to its previous value.

\*

ENTER MEDIAN STO Enters the program, then stores it in

MEDIAN.

**Checksum:** # 19502d **Bytes:** # 129.5

**Example.** Calculate the median of the following data.

There are two columns of data, so MEDIAN will return a two-element vector.

Enter the matrix.

MATRIX

18 ENTER 12 ENTER ▼

4 ENTER 7 ENTER

3 ENTER 2 ENTER

11 ENTER 1 ENTER

31 ENTER 48 ENTER

20 ENTER 17 ENTER

**ENTER** 

1: [[ 18 12 ] [ 4 7 ] [ 3 2 ] [ 11 1 ] [ 3 2 ]

Store the matrix in  $\Sigma DAT$ .

Calculate the median.

The medians are 14.5 for the first column and 9.5 for the second column.

#### **Expanding and Collecting Completely**

This section contains two programs:

- MULTI repeats a program until the program has no effect on its argument.
- EXCO calls MULTI to completely expand and collect an algebraic.

#### **MULTI (Multiple Execution)**

Given an object and a program that acts on the object, MULTI applies the program to the object repeatedly until the object is unchanged.

Arguments	Results
2: object	2:
1: « program »	1: resulting object

#### Techniques.

- DO... UNTIL... END (indefinite loop). The DO clause contains the steps to be repeated; the UNTIL clause contains the test that determines whether to repeat both clauses again (if false) or to exit (if true).
- Programs as arguments. Although programs are commonly named and then executed by calling their names, programs can also be put on the stack and used as arguments to other programs.
- Evaluation of local variables. The program argument to be executed repeatedly is stored in a local variable. It's convenient to store an object in a local variable when you don't know beforehand how many copies you'll need.

Recall from page 98 that an object stored in a local variable is simply put on the stack when the local variable is evaluated. *MULTI* uses the local variable name to put the program argument on the stack and then executes EVAL to execute the program.

Program:	Comments:
«	
→ p	Creates a local variable $p$ containing the program from level 1.
«	Begins the defining procedure (a program) for the local variable structure.

DO Begins the DO clause.

DUP Makes a copy of the object, now in

level 1.

P EVAL Applies the program to the object,

returning a new version.

DUP Makes a copy of the new version of

the object.

ROT Moves the old version to level 1.

UNTIL Begins the UNTIL clause.

SAME Tests whether the old version and the

new version are the same.

END Ends the UNTIL clause.

Ends the defining program.

Ends the program.

ENTER [] MULTI STO Puts the program on the stack, then

stores it in MULTI.

Checksum: # 34314d

Bytes: 56

MULTI is demonstrated in the next programming example.

#### **EXCO (Expand and Collect Completely)**

Given an algebraic object, EXCO executes EXPAN repeatedly until the algebraic doesn't change, then executes COLCT repeatedly until the algebraic doesn't change. In some cases the result will be a number.

Arguments	Results
1: 'algebraic'	1: 'algebraic'
1: 'algebraic'	1: z

#### Techniques.

■ Subroutines. EXCO calls the program MULTI twice. It is more efficient to create program MULTI and simply call its name twice than write each step in MULTI two times.

#### **Required Programs.**

■ MULTI (page 569) repeatedly executes the programs that EXCO provides as arguments.

Program:	Comments:
*	
« EXPAN »	Puts a program on the stack as the level 1 argument for <i>MULTI</i> . The program executes the EXPAN command.
MULTI	Executes EXPAN until the algebraic object doesn't change.
« COLCT »	Puts another program on the stack for <i>MULTI</i> . The program executes the COLCT command.
MULTI	Executes COLCT until the algebraic object doesn't change.
<b>»</b>	
ENTER (1 EXCO STO	Puts the program on the stack, then stores it in EXCO.

**Checksum:** # 48008d **Bytes:** 65.5

**Example.** Expand and collect completely the expression:

$$3x(4y+z)+(8x-5z)^2$$

Enter the expression.

1 3 × X × 1 1 1 4 × Y + Z ► + 1 1 1 8 × X - 5 × Z 1 2 ENTER 1: '3\*X\*(4\*Y+Z)+(8\*X-5 \*Z)^2' \*\* 01 EGHT EGISP FIE1 FIE2

Select the VAR menu and start the program.

VAR EXCO

Expressions with many products of sums or with powers can take many iterations of EXPAN to expand completely, resulting in a long execution time for EXCO.

# Finding the Minimum or Maximum Element of an Array

This section contains two programs that find the minimum or maximum element of an array:

- MNX uses a DO ... UNTIL ... END (indefinite) loop.
- MNX2 uses a FOR ... NEXT (definite) loop.

### MNX (Finding the Minimum or Maximum Element of an Array — Technique 1)

Given an array on the stack, MNX finds the minimum or maximum element in the array.

Arguments	Results
1: [[array]]	2: [[array]] 1: z (maximum element of array)
1: [[array]]	2: [[array]] 1: z (minimum element of array)

- DO... UNTIL... END (indefinite loop). The DO clause contains the sort instructions. The UNTIL clause contains the system-flag test that determines whether to repeat the sort instructions.
- User and system flags for logic control:
  - User flag 10 defines the sort: When flag 10 is set, MNX finds the maximum element; when flag 10 is clear, it finds the minimum element. You determine the status of flag 10 at the beginning of the program.
  - System flag -64, the Index Wrap Indicator flag, determines when to end the sort. While flag -64 is clear, the sort loop continues. When the index invoked by GETI wraps back to the first array element, flag -64 is automatically set, and the sort loop ends.
- Nested conditional. An IF... THEN... END conditional is nested in the DO... UNTIL... END conditional—it determines:
  - Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
  - The sense of the comparison of elements (either < or >) based on the status of flag 10.
- Custom menu for making a choice. MNX builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled MAX, sets flag 10. Key 2, labeled MIN, clears flag 10.

■ Logical function. MNX executes XOR (exclusive OR) to test the combined state of the relative value of the two elements and the status of flag 10.

Program:	Comments:
«	
<b>€</b>	Begins the defining list for the option menu.
( "MAX" « 10 SF CONT » ) ( "MIN" « 10 CF CONT » )	Builds menu keys MAX to set flag 10 and continue program execution, and MIN to clear flag 10 and continue program execution.
>	Ends the defining list for the temporary option menu.
TMENU "Sort for MAX or MIN?" PROMPT	Displays the temporary menu and a prompt message.
1 GETI	Gets the first element of the array.
DO	Begins the DO loop.
ROT ROT	Puts the index and the array in levels 1 and 2.
GETI 4 ROLL DUP2	Gets the new array element, moves the current minimum or maximum array element from level 4 to level 1. Then copies both elements.
IF	Begins the conditional.
> 10 FS? XOR	Tests the combined state of the relative value of the two elements and the status of flag 10.
THEN	If the new element is either less than the current maximum or greater than the current minimum

SWAP ... swaps the new element into level

1.

END Ends the conditional.

DROP Saves the current minimum or

maximum and drops the other

element off the stack.

UNTIL Begins the UNTIL clause.

-64 FS? Tests if flag -64 is set. If flag -64 is

clear, executes the DO clause again.

END If flag -64 is set, ends the loop.

SWAP DROP 0 MENU Swaps the index to level 1, then

drops it off the stack. Restores the

last menu.

\*

ENTER MNX STO Enters the program, then stores it in

MNX.

**Checksum:** # 57179d **Bytes:** 210.5

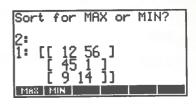
**Example.** Find the maximum element of the following matrix:

Enter the matrix.

MATRIX
12 ENTER 56 ENTER ▼
45 ENTER 1 ENTER
9 ENTER 14 ENTER
ENTER

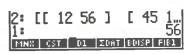
1: [[ 12 56 ] [ 45 1 ] [ 9 14 ]] MINE UST 01 MONT BOUSE FIEL Select the VAR menu and execute MNX.

VAR MNX



Find the maximum element.

MAX



### MNX2 (Finding the Minimum or Maximum Element of an Array—Technique 2)

Given an array on the stack, MNX2 finds the minimum or maximum element in the array. MNX2 uses a different approach than MNX; it executes OBJ— to break up the array into individual elements on the stack for testing, rather than executing GETI to index through the array.

Arguments	Results
1: [[array]]	2: [[array]] 1: z (maximum element of array)
1: [[array]]	2: [[array]] 1: z (minimum element of array)

- FOR... NEXT (definite loop). The initial counter value is 1. The final counter value is nm 1 where nm is the number of elements in the array. The loop-clause contains the sort instructions.
- User flag for logic control. *User* flag 10 defines the sort: When flag 10 is set, *MNX2* finds the maximum element; when flag 10 is clear, it finds the minimum element. *You* determine the status of flag 10 at the beginning of the program.
- Nested conditional. An IF ... THEN ... END conditional is nested in the FOR ... NEXT loop it determines:

- Whether to maintain the current minimum or maximum element, or make the current element the new minimum or maximum.
- The sense of the comparison of elements (either < or >) based on the status of flag 10.
- Logical function. MNX2 executes XOR (exclusive OR) to test the combined state of the relative value of the two elements and the status of flag 10.
- Custom menu for making a choice. MNX2 builds a custom menu that lets you choose whether to sort for the minimum or maximum element. Key 1, labeled MAX, sets flag 10. Key 2, labeled MIN, clears flag 10.

Program:	Comments:
« har gets process of members and	
•	Begins the defining list for the temporary option menu.
<pre>{ "MAX" « 10 SF CONT » } { "MIN" « 10 CF CONT » }</pre>	Builds menu keys MAX to set flag 10 and continue program execution, and MIN to clear flag 10 and continue program execution.
	Ends the defining list for the option menu.
TMENU "Sort for MAX or MIN?" PROMPT	Displays the temporary menu and a prompting message.
DUP	Copies the array.
OBJ→	Returns the individual array elements to levels 2 through $nm+1$ , and returns the list containing $n$ and $m$ to level 1.
1 Selection were a respective	Sets the initial counter value.
SWAP OBJ>	Converts the list to individual elements on the stack.

DROP \* 1 -

Drops the list size, then calculates the final counter value (nm - 1).

FOR n

Starts the FOR ... NEXT loop.

DUP<sub>2</sub>

Saves the array elements to be tested (initially the last two elements). Establishes the last array element as the current minimum or maximum.

IF

Begins the conditional.

> 10 FS? XOR

Tests the combined state of the relative value of the two elements and the status of flag 10.

THEN

If the new element is either less than the current maximum or greater than the current minimum ...

SWAP

... swaps the new element into level

1.

END

Ends the conditional.

DROP

Saves the current minimum or maximum (and drops the other element off the stack).

NEXT

Ends the FOR ... NEXT loop.

0 MENU

Restores the last menu.

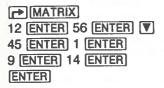
>

ENTER MNX2 STO

Enters the program, then stores it in MNX2.

Checksum: Bytes: # 12277d 200.5 **Example.** Use MNX2 to find the minimum element of the matrix from the previous example:

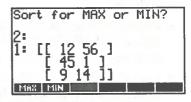
Enter the matrix.





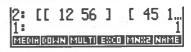
Select the VAR menu and execute MNX2.

VAR MNX2



Find the minimum element.

MIN



#### **Verification of Program Arguments**

The two utility programs in this section verify that the argument to a program is the correct object type.

- NAMES verifies that a list argument contains exactly two names.
- VFY verifies that the argument is either a name or a list containing exactly two names. It calls NAMES if the argument is a list.

You can modify these utilities to verify other object types and object content.

### NAMES (Does the List Contain Exactly Two Names?)

If the argument for a program is a list (as determined by VFY), NAMES verifies that the list contains exactly two names. If the list does not contain exactly two names, an error message is displayed in the status area and program execution is aborted.

Arguments	Results
1: { valid list }	1:
1: { invalid list }	status-area error message 1:

- Nested conditionals. The outer conditional verifies that there are two objects in the list. If there are two objects, the inner loop verifies that they are both names.
- Logical functions. NAMES uses the AND command in the inner conditional to determine if both objects are names and the NOT command to display the error message if they are not both names.

Program:	Comments:
«	
IF	Starts the outer IFTHENELSEEND structure.
OBJ÷	Returns the $n$ objects in the list to levels 2 through $(n + 1)$ , and returns the list size $n$ to level 1.
DUP	Copies the list size.
2 SAME	Tests if the list size is 2.

If the list size is 2... THEN ... moves the objects to levels 1 and DROP 2. Begins the inner IF...THEN...END IF structure. Tests if the first object is a name. If TYPE 6 SAME so, returns a true result (1). If not, returns a false result (0). Moves the second object to level 1, SWAP TYPE 6 SAME then tests if it is a name. AND If both results are true, returns a true result (1). If either or both results are false, returns a false result (0).Returns the opposite result. NOT

THEN

If the opposite result is true (if the objects are not both names) ...

"List needs two names" ... displays an error message and aborts program execution.

END Ends the inner conditional.

ELSE If the list size is not 2...

DROPN ... drops the list size, displays an "Illegal list size" error message, and aborts program execution.

END Ends the outer conditional.

ENTER [] NAMES STO Enters the program and stores it in NAMES.

**Checksum:** # 40666d **Bytes:** 141.5

NAMES is demonstrated in program VFY.

#### **VFY (Verify Program Argument)**

Given an argument on the stack, VFY verifies that the argument is either a name or a list that contains exactly two names.

Arguments	Results
1: 'name'	1: 'name'
1: { valid list }	1: { valid list }
1: { invalid list }	status—area error message 1: { invalid list }
1: invalid object	status-area error message 1: invalid object

#### Techniques.

- Utility programs. VFY by itself has little use. However, it can be used (with minor modifications) by other programs to verify that specific object types are valid arguments.
- CASE... END (case structure). VFY uses a case structure to determine if the argument is a list or a name.
- Structured programming. If the argument is a list, VFY calls NAMES to verify that the list is valid.
- Local variable structure. VFY stores its argument in a local variable so that it may be passed to NAMES if necessary.
- Logical operator. VFY uses NOT to display an error message.

#### Required Programs.

■ NAMES (page 580) verifies that a list argument contains exactly two names.

Comments: Program: 4 Saves the original argument. DUP Removes any tags from the DTAG argument for subsequent testing. Stores the argument in local variable → arg arg. Begins the defining procedure (a « program) for the local variable structure. Begins the case structure. CASE Tests if the argument is a list. arg TYPE 5 SAME If the argument is a list ... THEN ... puts the argument back on the arg NAMES stack, and calls NAMES to verify that the list is valid. Ends the first case. (If the first case END was true, leaves the case structure. If the first case was false, goes to the next case.) arg TYPE 6 SAME NOT Tests if the argument is a name, then inverts the test result. If the argument is not a name (and THEN not a list) ... ... displays an error message and "Not name or list" aborts program execution. DOERR

END Ends the second case.

END

Ends the case structure.

Ends the defining procedure.

>

ENTER 'VFY STO

Enters the program, then stores it in

VFY.

**Checksum:** # 14621d **Bytes:** 135.5

**Example. Part 1.** Execute VFY to test the validity of the name argument PAT.

Put the name PAT on the stack. Select the VAR menu and execute VFY.

PAT ENTER

VAR

VFY

1: 'PAT'

The argument is valid and is simply returned to the stack.

**Part 2.** Execute VFY to test the validity of the list argument  $\{PAT DIANA \ TED \}$ .

Put the names DIANA and TED on the stack. Convert the three names now on the stack to a list.

DIANA ENTER

TED ENTER

3 PRG OBJ →LIST

1: ( PAT DIANA TED )

Execute VFY. Since the list contains too many names, the error message is displayed and program execution is aborted.

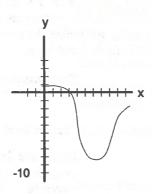
VAR YFY

Illegal list size
4:
3:
2:
1: { PAT DIANA TED }

#### **Bessel Functions**

The real and imaginary parts of the Bessel function  $J_n(xe^{\frac{3n}{4}})$  are denoted  $Ber_n(x)$  and  $Bei_n(x)$ . When n=0,

Ber(x) = 1 - 
$$\frac{(x/2)^4}{2!^2}$$
 +  $\frac{(x/2)^8}{4!^2}$  - ...



User-defined function BER calculates Ber(x) to 12 significant digits.

Arguments	Results
1: Z	1: Ber(z)

#### Techniques.

■ Local variable structure. BER consists solely of a local variable structure and so has two properties of a user-defined function; it takes numeric or symbolic arguments from the stack or in algebraic syntax. Because BER uses a FOR ... STEP loop, its defining procedure is a program. (Loop structures are not allowed in algebraic expressions.) Therefore, unlike a user-defined function, BER is not differentiable.

- FOR ... STEP loop (definite loop with counter). Successive terms in the series are calculated with a counter-controlled loop. When the new term does not change the series value within the 12-digit precision of the calculator, the loop ends. The final counter value  $(9.0 \times 10^{499})$  ensures that enough terms will be calculated.
- Nested conditional. The IF ... THEN ... ELSE ... END conditional within the definite loop sets the step value n for the loop counter. As long as the newly calculated series value does not equal the old series value, the step value n is set to 2. When the new series value does equal the old series value, the step value is set to a number larger than the final value of the counter, ending the definite loop. In essence, the nested conditional makes the outer loop work like a DO ... UNTIL ... END (indefinite) loop.

	_			
«				

Program:

#### Comments:

«	
→ ×	Creates local variable x.
«	Begins the defining procedure (a program) for the local variable structure.
1	Writes the first term of the series.
2 9.E499	Sets the counter for the FOR STEP loop.
FOR j	Begins the loop.
DUP	Saves the current value of the series (initially 1).
'(-1)^(j/2)* (x/2)^(2*j) /SQ(j!)' EVAL	Calculates the next term of the series.
The second of the second	Adds the next term to the current value of the series to calculate the new value of the series.

IF

Begins the conditional.

DUP ROT #

Tests if the new series value is not

equal to the old series value.

THEN

If the new and old values are not

equal ...

... specifies n = 2.

ELSE

If the new and old terms are equal

(to 12-digit precision) ...

9.1E499

... specifies n = 9.1E499

END

Ends the conditional.

STEP

Specifies the step value based on the

conditional.

\*

Ends the defining procedure.

\*

[ENTER] [] BER [STO]

Enters the program, then stores it in

BER.

Checksum:

# 872d

**Bytes:** 

148

Example. Calculate Ber(3).

VAR

3 BER

Calculate Ber(2) in algebraic syntax.

BER (1) 2 **EVAL** 

BER SINTP SETTS

## Animation of Successive Taylor's Polynomials

This section contains three programs that manipulate graphics objects to display a sequence of Taylor's polynomials for the sine function.

- **SINTP** draws a sine curve, and saves the plot in a variable.
- SETTS superimposes plots of successive Taylor's polynomials on the sine curve plot from SINTP, and saves each graphics object in a list.
- TSA displays in succession each graphics object from the list built in SETTS.

### **Drawing a Sine Curve and Converting It to a Graphics Object**

SINTP draws a sine curve, returns the plot to the stack as a graphics object, and stores that graphics object in a variable.

Arguments	Results
1:	1:

#### Techniques.

Programmatic use of PLOT commands to build and display a graphics object.

Program:	Comments:
*	
'X' PURGE 'SIN(X)' STEQ	Makes $X$ a formal variable, then stores the expression for $\sin x$ in $EQ$ .
-2 2 YRNG	Sets the y-axis display range.
ERASE DRAW	Erases <i>PICT</i> , then plots the expression.

PICT RCL 'SINT' STO

Returns the resultant graphics object to the stack and stores it in SINT.

3

ENTER | SINTP STO

Stores the program in SINTP.

Checksum:

# 61373d

**Bytes:** 78.5

#### **Superposition of Successive Taylor's Polynomials**

SETTS superimposes successive Taylor's polynomials on a sine curve and stores each graphics object in a list.

Arguments	Results
1:	1:

- Structured programming. SETTS calls SINTP to build a sine curve and convert it to a graphics object.
- FOR...STEP (definite) loop. SETTS calculates successive Taylor's polynomials for the sine function in a definite loop. The loop counter serves as the value of the order of each polynomial.
- Programmatic use of PLOT commands. SETTS draws a plot of each Taylor's polynomial.
- Manipulation of graphics objects. SETTS converts each Taylor's polynomial plot into a graphics object. Then it executes + to combine each graphics object with the sine curve stored in SINT, creating nine new graphics objects, each the superposition of a Taylor's polynomial on a sine curve. SETTS then puts the nine new graphics objects, and the sine curve graphics object itself, in a list.

#### Program:

Comments:

4

SINTP

17 1 FOR ×

× 'X' DUP SIN SWAP ROT TAYLR STEQ ERASE DRAW

PICT RCL SINT +

-2 STEP

SINT 10 →LIST 'TSL' STO

ENTER SETTS STO

**Checksum:** # 5841d **Bytes:** 136.5

Plots a sine curve and stores the graphics object in SINT.

For each value of local variable  $x ext{...}$ 

... plots the Taylor's polynomial for the sine curve (where x is the order of the polynomial).

Returns the plot to the stack as a graphics object and executes + to superimpose the Taylor series on the sine curve stored in SINT.

Decrements the loop counter (the order of the Taylor's polynomial) by 2 and repeats the loop.

Puts the sine curve graphics object on the stack, then builds a list that contains that graphics object and the nine graphics objects created in the FOR... STEP loop. Stores the list in TSL.

Stores the program in SETTS.

#### **Animation of Taylor's Polynomials**

TSA displays in succession each graphics object created in SETTS.

Arguments	Results	
1:	1:	

- Passing a global variable. Because SETTS takes a long time to execute (approximately six minutes), TSA does not call SETTS. Instead, you must first execute SETTS to create the global variable TSL containing the list of graphics objects. TSA simply executes that global variable to put the list on the stack.
- FOR ... NEXT (definite loop). TSA executes a definite loop to display in succession each graphics object from the list.

Program:	Comments:
« data mental trade out	
TSL	Puts the list TSL on the stack.
OBJ→	Puts the 10 graphics objects from the list and the list count on the stack.
1 SWAP FOR s	For s from 1 to 10
ERASE →LCD 1 WAIT	clears the display, converts the level-1 graphics object to a display
	image, and shows it for one second.
NEXT	
»	
ENTER 1 TSA STO	Stores the program in TSA.

**Checksum:** # 39562d

Bytes: 51

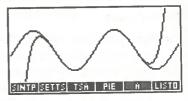
**Example.** Execute SETTS and TSA to build and display in succession a series of Taylor's polynomial approximations of the sin function.

Set Radians mode. Execute SETTS to build the list of graphics objects. SETTS takes about six minutes to execute. Execute TSA to display each plot in succession. The display shows TSA in progress.

← RAD (if necessary)

VAR SETTS

TSR



### **Programmatic Use of Statistics and Plotting**

Program PIE prompts for single variable data, stores that data in the statistics matrix  $\Sigma DAT$ , then draws a labeled pie chart that shows each data point as a percentage of the total.

X (1)	Arguments	R	esults
1:		1:	

- Programmatic use of PLOT commands. PIE executes XRNG and YRNG to define x- and y-axis display ranges in user units, executes ARC to draw the circle, and LINE to draw individual slices.
- Programmatic use of matrices and statistics commands.
- Manipulation of graphics objects. *PIE* recalls *PICT* to the stack and executes GOR to merge the label for each slice with the plot.
- FOR ... NEXT (definite) loop. Each slice is calculated, drawn and labeled in a definite loop.

- CASE... END structure. To avoid overwriting the circle, each label is offset from the midpoint of the arc of the slice. The offset for each label depends on the position of the slice in the circle. The CASE... END structure assigns an offset to the label based on the position of the slice.
- Preservation of current calculator flag status. Before specifying Radians mode, *PIE* saves the current flag status in a local variable, then restores that status at the end of the program.
- Temporary menu for data input.

Comments:
Recalls the current flag status and stores it in variable flags.
Sets Radians mode.
Begins the defining list for the input menu.
Defines key 1. Key 1 executes $\Sigma$ + to store each data point in $\Sigma DAT$ .
Defines keys 2 and 3. Key 3 clears $\Sigma DAT$ .
Defines keys 4, 5, and 6. Key 6, labeled DRAW continues program execution after data entry.
Ends the defining list.
Displays the temporary menu.
Prompts for inputs. The is the calculator's representation of the character ( ) after the program has been entered on the stack.

Erases the current <i>PICT</i> and sets plot parameters.
Displays "drawing" message.
Executes ARC to draw the circle.
Displays the empty circle.
Recalls the statistics data matrix, computes totals, and calculates the proportions.
Converts the proportions to percentages.
Stores the percentage matrix in prents.
Multiplies the proportion matrix by $2\pi$ .
Stores the proportions in <i>prop</i> and initializes <i>angle</i> to 0.
Sets up start and finish for FORNEXT loop.
Begin FOR clause.
Puts the center of the circle on the stack and gets the xth value from the proportion matrix.
Computes the endpoint and draws the line for the xth slice.

PICT RCL

Recalls PICT to the stack.

angle prop x GET

2 / - DUP

COS LASTARG SIN R→C 26 \* (66,32) + For labeling the slice, computes the midpoint of the arc of the slice.

SWAP DUP

Starts the CASE...END structure to determine the offset value for the label.

1.5 ≤

THEN

From 0 to 1.5 radians ...

DROP

... doesn't offset the label.

END

DUP 4.4 4

THEN

From 1.5 to 4.4 radians ...

DROP 15 -

... offsets the label 15 user units left.

END

5 <

THEN

From 4.4 to 5 radians ...

(3,2) +

... offsets the label 3 units right and

2 units up.

**END** 

**END** 

pronts x GET

Gets the xth value from the

percentage matrix.

1 RND

Rounds the percentage to one

decimal place.

⇒STR "%" +

Converts the percentage to a string and adds % to the string.

1 →GROB Converts the string to a graphics

object.

GOR DUP PICT STO Adds the label to the plot and stores

the new plot.

⇒LCD Displays the plot with the new slice

and label.

NEXT

C 3 PVIEW Displays the finished plot.

\*

>>

flags STOF Restores the original flag status.

» 2 MENU Displays the VAR menu. (Note that

the user must first press ATTN to

clear the plot.)

>

ENTER PIE STO Enters the program and stores it in

PIE.

**Checksum:** # 8706d **Bytes:** 758.5

**Example.** The fruit inventory at Joe's grocery includes 983 oranges, 416 apples, and 85 bananas. Draw a pie chart to show each fruit's percentage of total inventory.

Start PIE.

VAR PIE



Clear the current statistics data. (The prompt is removed from the display.) Key in the new data and draw the pie chart.

CLEAR 983 SLICE 416 SLICE 85 SLICE DRAW



#### **Animation of a Graphical Image**

Program WALK shows a man walking across the display. It animates this custom graphical image by incrementing the image position in a loop structure.

Arguments		Results	
1:		1:	

- Use of a custom graphical image in a program. (Note that the programmer derives the full information content of the graphical image before writing the program by building the image interactively in the Graphics environment and then returning it to the command line.)
- FOR...STEP definite loop to animate the graphical image. The ending value for the loop is MAXR. Since the counter value cannot exceed MAXR, the loop executes indefinitely.

#### Program:

Comments:

4

GROB 9 15 E300 140015001C001400E300 8000C110AA0094009000 4100220014102800 Puts the graphical image of the man in the command line. (Note that the hexadecimal portion of the graphics object is a continuous integer E300...2800. The linebreaks do not represent spaces.)

→ man

Creates local variable man containing the graphics object.

\*

ERASE ( # 0d # 0d )
PYIEW

Clears PICT, then displays it.

{ # 0d # 25d } PICT OVER man GXOR Puts the first position on the stack and turns on the first image. This readies the stack and *PICT* for the loop.

5 MAXR FOR i

Starts FOR...STEP loop to generate horizontal coordinates indefinitely.

i 131 MOD R→B

Computes the horizontal coordinate for the next image.

# 25d 2 +LIST

Specifies a fixed vertical coordinate. Puts the two coordinates in a list.

PICT OVER man GXOR

Displays the new image, leaving its coordinates on the stack.

PICT ROT man GXOR

Turns off the old image, removing its coordinates from the stack.

5 STEP

Increments the horizontal coordinate by 5.

\*

..

ENTER ' WALK STO

Stores the program in WALK.

Checksum:

# 4342d

Bytes:

236.5

**Example.** Send the man out for a long walk.

Select the VAR menu and execute WALK.

VAR WALK



When he tires, press ATTN to take him home (and end the program).

#### en able to little feet of special state

nesse in the English Color of the Color of t

to the consequent to the West with the terminal point that the prophiles the consequent to the terminal transfer the consequent to the con

1. The almost group arts exect?

Course was a few or a second of the second o

Fig. on the poster characteristic value of the sentence of the

g ampang pilot sa kang sanakan ka

d en e la lain

s<sub>p</sub>, estima y fosco son med a sessonatana seg<sub>p</sub>, may a sessonatana es lo coses

2 1 201

water's a set on the set of the s

### Part 5

# Printing, Data Transfer, and Plug-Ins

# **Printing**



This chapter describes how to use your HP 48 with an HP 82240B Infrared Printer, with an HP 82240A infrared printer, and with printers that connect to the serial port.

# Printing with an HP 82240B Printer

You can send information from your HP 48 to an HP 82240B Infrared Printer via the infrared port. Refer to the printer manual for instructions about how to operate the printer and how to position the printer relative to the HP 48.

#### **PRINT Commands**

Keys	Programmable Command	Description
ON MTH		When ON and MTH are pressed simultaneously and then released, the current display is printed.
PRINT	PR1	Prints the object in level 1.
PRINT:		
	PR1	Prints the object in level 1.
PRST	PRST	Prints all objects on the stack starting with the object in the highest level.
PRSTC	PRSTC	Prints all objects on the stack in compact form, starting with the object in the highest level.
PRLCD	PRLCD	Prints the current display.
PRVAR	PRVAR	Searches the current path for the specified variables, and prints the name and contents of each variable. The variables are specified either by name or in a list in level 1.
CR	CR	Causes printer to do a carriage- return/line-feed, printing the contents, if any, of the printer buffer.
DELAY	DELAY	Sets the delay time, ≤ 6.9 seconds, between sending lines of information to the printer.
OLDPR	OLDPRT	Remaps the HP 48 character set to the HP 82240A Infrared Printer.

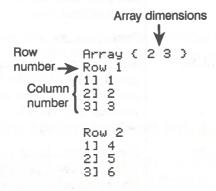
#### **Print Formats**

Multiline objects can be printed in *multiline* format or *compact* format. Multiline printer format is similar to multiline display format, with the following exceptions:

- Strings and names that are more than 24 characters long are continued on the next printer line.
- The real and imaginary parts of complex numbers are printed on separate lines if they don't fit on the same line.
- Arrays are printed with a numbered heading for each row and with a column number before each element. For example, the 2×3 array

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

would be printed like this:



Compact printer format is the same as compact display format: Multiline objects are truncated and appear on one line only.

The PRSTC command prints the stack in compact form. All other print commands print in multiline format.

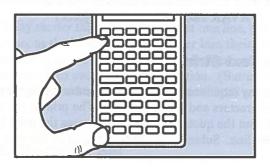
#### **Basic Printing Commands**

**Printing the Display.** To print an image of the display under any condition without using the PRINT menu:

- 1. Press and hold ON.
- 2. Press and release MTH (the key with "PRINT" written above it).
- 3. Release ON.



A low-battery condition may result in consistent failure of the ONMTH printing procedure. If you notice consistent failure, replace your calculator batteries to remedy the situation.



The PRLCD command ( PRINT PRLCD) also prints an image of the display.

<sup>\*</sup> These keystrokes use the current DELAY setting. Also, if you are printing to the serial port to capture graphics data on your printer, the serial port must be open (the OPENIO command) before these keystrokes are executed.

Printing the Contents of Level 1 of the Stack. PR1 (PRINT) prints the contents of level 1 in multiline printer format. All objects except strings are printed with their identifying delimiters. Strings are printed without their "delimiters. PR1 can be executed also by pressing PRINT.

Printing the Stack. PRST (PRINT PRST) prints all objects on the stack, starting with the object in the highest level, in multiline printer format (except for graphics objects, which print the same as they are displayed).

PRSTC ( PRINT PRSTC) prints all objects on the stack, starting with the object in the highest level, in compact printer format.

Printing Variables. PRVAR ( PRINT PRVAR) searches the current path for the variables that you have specified, and prints the name and contents of each variable in multiline printer format. PRVAR takes one argument from the stack: either one name or a list containing one or more names. (PRVAR also prints backup objects.)

#### **Printing a Text String**

You can print any sequence of characters by entering a string object that contains the characters and executing PR1. The printer prints the characters without the quotation marks and leaves the print head at the end of the print line. Subsequent printing begins on the next line.

#### **Printing a Graphics Object**

Like other objects, you can print a graphics object either by putting the graphics object in level 1 and executing PR1, or, if the graphics object is stored in a variable, by entering the variable name and executing PRVAR. Graphics objects wider than 166 dot columns are printed in 166-column wide segments down the paper, separated by a dashed line. For example, a 350-column wide graphics object would be printed in two 166-column segments and one 18-column segment.

#### **Double Space Printing**

To select double-space printing (one blank line between lines), set flag -37. To return to single-space printing, clear flag -37.

#### **Setting the Delay**

The DELAY command lets you specify how long the HP 48 waits between sending lines of information to the HP 82240B Infrared Printer. DELAY takes a real number from level 1 that specifies the delay time in seconds. If you do not specify a delay, it is automatically set to 1.8 seconds. The maximum delay is 6.9 seconds.

A shorter delay setting can be useful when the HP 48 sends multiple lines of information to your printer (for example, when printing a program). To optimize printing efficiency, set the delay just longer than the time the printhead requires to print one line of information.

If you set the delay *shorter* than the time to print one line, you may lose information. Also, as the batteries in the printer lose their charge, the printhead slows down, and, if you have previously decreased the delay, you may have to increase it to avoid losing information. (Battery discharge will not cause the printhead to slow to more than the 1.8 second default delay setting.)

#### The HP 48 Character Set

The table in appendix C lists each HP 48 character and its corresponding character code. Most of the characters in the table can be directly typed into the display from the Alpha keyboard. For example, to display \$, type a. (The Alpha keyboard is presented in chapter 2.) Any character in the table can be displayed by typing its corresponding character code and then executing the CHR command. The syntax is char# CHR. Certain characters in the table in appendix C are not on the Alpha keyboard. To display one of these characters, you must type its character code and execute CHR.

The HP 82240B Infrared Printer can print any character from the HP 48 character set.

# Sending Escape Sequences and Control Codes

You can select various printer modes by sending escape sequences to the printer. An escape sequence consists of the escape character—character 27—followed by additional characters. When the printer receives an escape sequence, it switches into the selected mode. The escape sequence itself isn't printed.

Printer owner's manuals generally describe the escape sequences and control codes recognized by the printer.

Use CHR and + to create escape sequences and use PR1 to send them to the printer.

**Example.** These characters send information to the HP 82240B printer to turn on Underline mode, underline the string HELLO, and then turn off Underline mode:

27 CHR 251 CHR + "HELLO" + 27 CHR + 250 CHR + PR1

# **Accumulating Data in the Printer Buffer**

You can print any combination of text, graphics, and objects on a single print line by accumulating data in the printer's buffer.

Normally, each print command completes data transmission by automatically executing the CR (carriage right) command, which tells the printer to do a carriage-return/line-feed. Then the printer prints the data currently in its buffer and leaves the print head at the right end of the print line.

You can disable the automatic execution of the CR command by setting flag -38, the Line-feed flag. Data from subsequent print commands is accumulated in the printer buffer and is printed only when you manually execute CR. When flag -38 is set, follow these three rules:

- Execute CR ( PRINT CR ) to print the accumulated data. (Alternately, send character 4 or character 10.)
- Print the data in the buffer before you accumulate more than 200 characters. Otherwise, the buffer fills up and subsequent characters are lost.

■ Allow time for the printer to print a line before sending more data.

The printer requires about 1.8 seconds per line.

Clear flag -38 to restore normal operation of the print commands.

# Printing with an HP 82240A Infrared Printer

You can use your HP 48 calculator with an HP 82240A Infrared Printer, executing the same print commands that you would use for an HP 82240B. However, the character set in the HP 82240A Infrared Printer does not match the HP 48 character set:h

- 24 characters in the HP 48 character set are not available in the HP 82240A Infrared Printer. (From the table in appendix C, these characters are numbers 129, 130, 143-157, 159, 166, 169, 172, 174, 184, and 185.) The HP 82240A prints a in substitution.
- Many characters in the extended character table (character codes 128 through 255) do not have the same character code. For example, the « character has code 171 in the HP 48 and code 146 in the HP 82240A Infrared Printer. If you want to use the CHR command to print extended characters with an HP 82240A Infrared Printer, first execute OLDPRT. OLDPRT adds a remap string to the PRTPAR variable, which changes the character code of each byte to match the codes in the HP 82240A Infrared Printer character table. (If you want to print a string containing graphics data, OLDPRT must not be in effect.)

If you executed OLDPRT to print with an HP 82240A Infrared Printer, and then want to print to an HP 82240B Infrared Printer, you should first purge the reserved variable *PRTPAR*. (You can first copy its contents to another variable if you want to save the settings for later use.) This resets the print parameters so that the character set matches the HP 82240B. (*PRTPAR* is described on page 611.)

# **Printing to the Serial Port**

You can print to a serial printer via the HP 48 serial port. Once the HP 48 is connected to the printer:

- 1. Set flag -34, the Printing Device flag.
- 2. Check that flag -33, the I/O Device flag is clear. (The default is clear.)\*
- 3. Set the HP 48 baud rate, parity, and translation code appropriately for your printer. These can be set using the I/O SETUP menu, described on page 617.
- 4. If your printer uses XON/XOFF handshaking, edit (or create) IOPAR to set transmit pacing ≠ 0. The reserved variable IOPAR is described on page 618.
- 5. If the number of characters that fit on one line on your printer is not 80, edit *PRTPAR* to contain the correct number as the third element in its list. (See the next section for information on *PRTPAR*.)
- **6.** If your printer requires a line termination sequence other than carriage-return/line-feed, edit *PRTPAR* to contain that sequence as the fourth element in its list. The reserved variable *PRTPAR* is described in the following section.

You can execute any of the print commands described in this chapter with a serial printer. However, note that:

- The maximum line length to print is specified in the reserved variable PRTPAR (described next).
- You cannot print a graphics object.

<sup>\*</sup> Setting both flags -33 and -34 would enable infrared serial data transmission. Printing with an HP 82240B Infrared Printer when these flags are set will not work—the HP 82240B would likely print blots.

#### The PRTPAR Variable

When you first print information with a command from the PRINT menu, the HP 48 automatically creates the *PRTPAR* variable. *PRTPAR* is a reserved variable containing a list that specifies how the HP 48 works with the printer. The list contains, in order, the following objects:

- A real number that specifies the delay time, in seconds. If you have not previously executed DELAY, the delay time is automatically set to 1.8 seconds in *PRTPAR*.
- A string that represents the current remapping of the HP 48 extended character set. The string can contain as many characters as you want to remap, with the first character in the string being the new character 128, the second being the new character 129, and so on. (Any characters outside the string length will not be remapped.) If you have not previously executed OLDPRT, the string is empty; if you have executed OLDPRT, the string contains the character remapping for the HP 82240A Infrared Printer.
- A real number that specifies the line length, in number of characters, for serial printing. This parameter does *not* affect infrared printing. The default is 80 characters.
- A string that represents the line termination method for serial printing. This parameter does *not* affect infrared printing. The default is *carriage-return/line-feed* (control characters 13 and 10).

You can edit any parameter in the list. The delay time, however, can be set more easily using the DELAY command: Enter the delay number (6.9 or less) on the stack and execute DELAY ( PRINT NXT DELAY).

# Transferring Data to and from the HP 48



#### This chapter covers:

- Transferring data from one HP 48 to another using the infrared port.
- Transferring data between the HP 48 and a computer using the serial port. (For this operation, you need the Serial Interface Kit appropriate for your computer. For more information, see your Hewlett-Packard dealer.)
- Other serial I/O operations.

The HP 48 uses *Kermit* file transfer protocol to transfer data and to correct transmission errors between two HP 48 calculators, or between an HP 48 and a computer. Kermit protocol was developed at the Columbia University Center for Computing Activities.

The calculator commands needed to accomplish Kermit data transfer are built into the HP 48. Therefore, you can transfer data from one HP 48 to another by simply lining up the two infrared ports and executing the proper commands, which are described in this chapter.

To transfer data to and from a computer, the computer must be running a program that implements Kermit protocol. Also, there must be a cable connecting the HP 48 and the computer. Details about the cable connection are covered later in this chapter. (Kermit protocol and a

special serial cable are required for this operation and are available from your Hewlett-Packard dealer as part of a Serial Interface Kit to match your computer.)

If you want additional information on Kermit protocol, a book by Frank da Cruz, KERMIT, A File Transfer Protocol, is available in many bookstores or can be ordered.

The HP 48 provides additional serial I/O commands for non-Kermit data transfers. These commands are for specialized I/O operations—for example, printing directly from the HP 48 to a serial printer.

# **Types of Data You Can Transfer**

The unit of information that is transferred using Kermit protocol is called a *file*. In the HP 48 world, a file can consist of:

- A named object (variable, backup object, etc.).
- An entire directory. When you transfer a directory, the contents of all the subdirectories under that directory are also transferred.
- All of user memory all the variables you've created, the user-key assignments, and the Alarm Catalog.

In all cases, a *copy* of the data is sent to the receiving device and stored as a file (variable) in the current directory.

When you transfer a directory or all of user memory between an HP 48 and a computer, the data is sent as a single file, and you cannot conveniently access the contents of the individual variables in that file. For this reason, a directory transfer to a computer should be done mainly for archiving purposes. When the purpose of a file transfer is to use the file at its destination (for example, to edit a program on your computer), you should transfer the contents of the individual variable. If you put the variable names in a list and use the SEND command to transfer the data, the variables can then be accessed individually.

<sup>\*</sup> da Cruz, Frank. 1987. KERMIT, A File Transfer Protocol. Bedford, MA: Digital Press.

When you transfer a directory from one HP 48 to another, it is installed in the destination machine as a normal directory. This means that it can be manipulated just like other directories and its variables are all accessible. Transferring a directory from one HP 48 to another is a good way to transfer a set of related objects—for instance, a set of programs, variables, printer configurations, etc.—all ready to be used together by the destination HP 48.

# The I/O Menu

The commands for Kermit protocol and serial operations are contained in the I/O menu. The serial commands are covered at the end of the chapter.

#### **Kermit Protocol Commands**

Keys	Programmable Command	Description	
<b>4</b> 1/0	pages 1 and 2):		
SEND SEND		Sends the contents of one or more variables to another device. SEND takes an argument from level 1—the variable name, or a list of names { name <sub>1</sub> name <sub>2</sub> }. (See the paragraph immediately following this table for more information.)	
REFEV	RECV	Tells the HP 48 to wait to receive a variable from another Kermit protocol device.	
SERVE	SERVER	Puts the HP 48 into Kermit Server mode. (Also executed by pressing	

### Kermit Protocol Commands (continued)

Keys	Programmable Command	Gets one or more variables from a server device. KGET takes an argument from level 1—the name of the requested variable, or a list of names {name1 name2 }. (See the paragraph immediately following this table for more information.)	
KGET	KGET		
FINIS	FINISH	Issues the Kermit FINISH command to a server device to terminate Server mode.	
	one second from the constant from the rock	Displays the SETUP menu for setting I/O parameters.	
	RECN	Same as RECV, except that it takes a name argument. The received file is stored using that name.	
PKT	PKT	Provides the ability to send a Kermit command "packet" to a server. It takes the packet data field as a string in level 2 and the packet type as a string in level 1. For example, "D" "G" PKT sends a "generic directory" command.	
KERR	KERRM	Returns the text of the most recent Kermit error.	
0)PENI	OPENIO	Opens the serial port using the I/O parameters in IOPAR.	
CLOSE	CLOSEIO	Closes the serial port, clears KERRM, and clears the input buffer.	

You can also use SEND and KGET to rename a variable when it's transferred by including a sublist for that variable in the main list. The first element in the sublist is the existing variable name and the second element is the new name. For example, executing the SEND command with the list { {name<sub>1</sub> name<sub>2</sub>} name<sub>3</sub> name<sub>4</sub> } as an argument would result in name<sub>3</sub> and name<sub>4</sub> being sent under their own names and name<sub>1</sub> being sent under the new name of name<sub>2</sub>.

#### **Local and Server Modes**

There are two Kermit protocol configurations for transferring data from an HP 48 to another HP 48 or computer:

- Local/Local. Both machines are controlled *locally* from their own keyboards, and Kermit commands can be issued by either machine. Data is transmitted by issuing a SEND command from the sender's keyboard and a RECV or RECN command from the receiver's keyboard.
- Local/Server. One machine is controlled *locally* and the other machine is a *server*. The server passively waits for instructions or data from the sender. A server:
  - Receives data when a sender executes a SEND command.
  - Transmits data when it receives a KGET command.
  - Ceases to be a server when it receives a FINISH command.

Local/Server mode is most useful when you wish to transfer a number of variables from different directories; the local device can issue repeated "send" or "get" commands to which the server responds.

# **Setting the I/O Parameters**

#### The SETUP Menu

Pressing SETUP displays the current I/O parameter settings and a menu for changing them. If the displayed settings are overwritten by the stack or other information, press [REVIEW] to redisplay them.

#### **SETUP Menu**

Keys	Programmable Command	Description
41/0	SETUP:	OT AND
IR/W		Switches between IR (infrared) and Wire (serial) modes. In IR mode, I/O output is directed to the infrared port. In Wire mode, I/O output goes to the serial port.
ASCII		Switches between ASCII and binary transmission modes (see page 629).
BAUD	BAUD	Steps through 1200, 2400, 4800, and 9600 baud. The default transfer rate is 9600 baud.
PARIT	PARITY	Steps through odd (1), even (2), mark (3), space (4), and no (0) parity. The default is no parity.

#### **SETUP Menu (continued)**

Keys	Programmable Command	Description
CKSM	CKSM	Steps through checksum (error detection) options. The CKSM set is the type of checksum requested when initiating a SEND. Choices are 1 (1-digit arithmetic checksum), 2 (2-digit arithmetic checksum), and 3 (3-digit cyclic redundancy check, or CRC).
	1 cd periode	The default is 3; IR transmissions should use 3.
TRAN	TRANSIO  3 cs seriale funcione e il ok.	Steps through the character translate code options. Choices are 0 (no translation), 1 (translate character 10 to characters 13 and 10), 2 (translate characters 128 through 159), or 3 (translate characters 128 through 255). The default is 1. (See page 626 for more information.)

The BAUD, PARITY, CKSM, and TRANSIO commands can be used in programs by preceding the command with the number representing the appropriate choice.

#### The IOPAR Variable

The reserved variable *IOPAR* stores the I/O parameters needed to establish a communications link with a computer. *IOPAR* contains a list consisting of these elements:

{ baud parity receive-pacing transmit-pacing checksum translate-code }

IOPAR is created in the HOME directory the first time you transfer data or open the serial port (OPENI). It is automatically updated whenever you change the settings using the commands in the I/O SETUP menu.

The Parity Setting. If the parity setting is positive, it is used on both transmit and receive. If it is negative, it is used only on transmit, and parity is not checked during receive. The menu key PARIT steps through only positive choices, but you can make the parity negative by putting the negative parity number on the stack, keying in the command PARITY, and pressing ENTER. You can also edit IOPAR, which contains the current I/O parameter settings, to make the parity element negative.

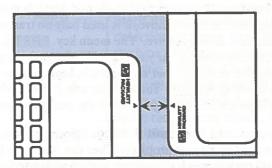
Receive Pacing and Transmit Pacing. Receive pacing and transmit pacing are not used by Kermit protocol. They can, however, be used in other serial I/O transfers—for instance, printing with a serial printer. A non-zero value for receive pacing causes the HP 48 to send an XOFF signal when its receive buffer is getting full, and then an XON signal when it can take more data. A non-zero value for transmit pacing causes the HP 48 to stop transmitting if it receives an XOFF signal and wait for an XON signal to continue. The default settings for both these IOPAR elements is 0, which means "don't send XON/XOFF signals, and ignore any that are received."

# **Transferring Data between Two HP 48's**

#### Before beginning the transfer:

- 1. On the sender, switch to the directory where the variables are located. Use the IO SETUP menu to set IR and binary transfer modes and to set the CKSM to 3.
- 2. On the receiver, use the IO SETUP menu to set IR transfer mode. Then, switch to the directory to which you want the data sent.

3. Line up the infrared ports by lining up the ▲ marks (near the Hewlett-Packard logo just above the display). The calculators should be no farther apart than 2 inches.



#### To transfer data using the local/local configuration:

- 1. On the receiver, do either of the following:
  - Execute RECV ( RECY ) to store the variable under the name given by the sender.
  - If you want to change the variable name, enter a new name and execute RECN ( NXT RECN ). When the object is received, it will be stored using that name.
- 2. On the sender, enter the name of the variable or directory to be sent and execute SEND ( SEND ). (For variables in the same directory, you can enter a list of variables and SEND them all at once.)
- 3. To transfer additional variables or lists of variables, repeat the previous two steps.

#### To transfer data using the local/server configuration:

1. On the HP 48 that will be the server, execute SERVER ( or I/O SERVE).

- 2. On the other, "locally controlled" HP 48:
  - To send a file to the server, enter the variable name and execute SEND ( SEND ). (To send the variable using a different name, or to send several variables from the same directory, use a list argument as described on page 616.)
- 3. To transfer additional variables or lists of variables, repeat step 2.
- 4. To end the session, execute FINISH ( ) on the locally controlled machine.

# Transferring Data between a Computer and the HP 48

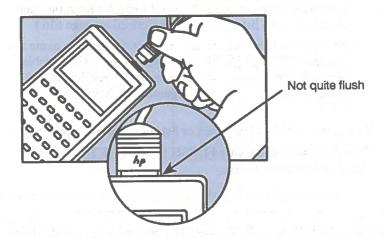
There are many reasons to transfer information between a computer and your HP 48—you might want to back up all of your calculator's user memory; you might want to edit a calculator program on your computer; or you might want to write a program on your computer and then run it on your calculator. Whatever the reason, the first step involves making a physical connection.

#### **Cable Connection**

Before transferring data between a computer and your calculator, you must connect the HP 48 to the computer via the serial cable in the Serial Interface Kit for your computer. (If you need information on what Serial Interface Kit is right for your computer, or if you don't have an Interface Kit, see your HP dealer.)

1. Connect the computer end of the serial cable to the serial port on the computer. (If you need instructions for this, consult your computer documentation.)

2. With the calculator right-side up and the HP logo on the cable connector facing up, connect the cable to your calculator. You should feel the connector lightly snap into place.



Note that when the cable is fully connected, the case around the connector is not quite flush with the calculator case.

## **Transferring Data**

#### Before beginning the transfer:

- 1. On the HP 48, display the I/O SETUP menu ( SETUP) and read the status message. If necessary:
  - Set Wire transmission mode by pressing IR/W.
  - Select ASCII or Binary transmission mode by pressing
     RSCII. (See page 629 for guidelines on selecting the mode to use.)
  - Set the HP 48 transfer rate by pressing BAUD until it matches the rate expected by the Kermit program running on the computer.
  - Set the HP 48 parity by pressing PRRIT until it matches the parity expected by the Kermit program running on the computer.
  - Set the checksum (CKSM)—type 1 is the fastest—and set the character translate code (TRRN). (See page 618 for guidelines on what translate code settings to use.)

- 2. On both the HP 48 and the computer, switch to the directory where the variables (files) are located and to the directory to which you want the variables (files) sent.
- 3. Open the HP 48 serial port by executing OPENIO ( IVO NXT OPENI). This step is not necessary for most connections, but it will prevent difficulties caused by the inability of certain devices to communicate with a closed port.
- 4. Run the program on the computer that implements Kermit protocol. If you are transferring data in binary mode, and if the Kermit program on the computer has a binary mode setting command, you should execute it on the computer.

#### To transfer data using the local/local configuration:

- 1. On the receiver, issue the "receive" command:
  - If the HP 48 is the receiver, execute RECV (► 1/0 RECV), or enter a variable name and execute RECN (► 1/0 NXT RECN).
  - If the computer is the receiver, issue the command on the computer to receive a file.
- 2. On the sender, issue the "send" command:
  - If the HP 48 is the sender, key in the argument (variable name or variable list as described on page 616) and execute SEND (♠)[/O SEND).
  - If the computer is the sender, issue the command on the computer to send a file.
- 3. To transfer additional variables or variable lists, repeat steps 1 and 2.
- 4. Optional: To conserve battery power, execute CLOSEIO ( USE) when finished.

DOCKING STATION VA SPENTA!

#### To transfer data using local/server configuration:

- 1. If your computer will be the server, make sure it is able to execute the Kermit "server" command.
- 2. Set server operation on the device that will act as server:
  - If the HP 48 is to act as server, execute SERVER ( or I/O SERV ).
  - If the computer is to act as server, execute the command on the computer to make it the server.
- 3. On the locally controlled device:
  - To send a file to the server, issue the appropriate "send" command. (See SEND on page 614 if the HP 48 is the sender.)
  - To receive a file from the server, issue the appropriate "get" command. (See KGET on page 615 if the HP 48 is the receiver.)
- 4. To transfer additional variables, repeat step 3.
- 5. To end the session, execute the "finish" command on the locally controlled machine. (If the HP 48 is locally controlled, press
- 6. Optional: To conserve battery power, execute CLOSEIO ( ) On the HP 48 when finished.

# **Backing Up All of HP 48 Memory**

The ARCHIVE and RESTORE commands provide the ability to back up all variables, user key assignments, and alarms in calculator memory onto your computer.

#### To backup all of user memory:

- 1. Follow the instructions in "Before Beginning the Transfer" on page 622.
- 2. Enter the object : IO: name, where name is the file name that will contain backed up memory. For example, : IO: AUG1 will back up memory into a file named AUG1.
- 3. Issue the Kermit RECEIVE command on the computer.
- 4. Execute ARCHIVE ( MEMORY NXT NXT HRCHI) to send the data to the PC. (Regardless of the ASCII/binary setting, ARCHIVE uses binary transmission.)

#### To copy backed up user memory into the HP 48:



Use the RESTORE command with care; restoring backed up user memory completely erases current user memory and replaces it with the backup copy.

- 1. Follow the instructions in "Before Beginning the Transfer" on page 622.
- 2. Transfer the computer file to the HP 48 the same way you transfer any other file.
- 3. Place the name of the file on the stack (for example, 'AUG1') and press RCL. This recalls Backup HOMEDIR to level 1.
- 4. Execute RESTORE ( MEMORY NXT NXT RESTO).

If you want your current flag settings archived when you back up all of memory, execute RCLF and store the result in a variable before you archive memory. Then, after you archive and restore memory, you can recall the contents of the variable and execute STOF to make the flag settings active again.

### **Character Translations (TRANSIO)**

The HP 48 character set contains certain characters that cannot be displayed using most computer software packages. These characters fall into two groups:

- Characters with "character numbers" in the range 128 through 159 cannot be displayed without special software designed to support the HP 48.
- Characters with character numbers in the range 160 through 255 can be displayed by computer software that supports the ISO 8859 character set.

The translate code lets you choose what happens to these characters when they are transmitted from the HP 48 to a computer. You set the translate code using the TRANSIO command. (See TRAN in the table on page 618 for a description of the four translate codes.)

The following table shows the conversions for many of the characters with numbers above 127. For characters not in the table, the conversion is to xxx, where xxx is the three-digit character number. This conversion makes it possible for you to use your computer editor to type and display these characters.

You can also use this conversion for characters in the table and for characters 0 through 127, making it easier to edit in control characters or in an escape sequence on your computer. The HP 48 will not generate the xxx sequences, but it will recognize them.

**I/O Character Translations** 

Char. Number	HP 48 Char.	PC Char.	Char. Number	HP 48 Char.	PC Char.
128	×.	\<)	147	ε	\Ge
129	$\overline{x}$	\x-	148	η	\Gn
130	♡	\.V	149	θ	\Gh
131	$\checkmark$	\v/	150	λ	\GI
132	ſ	\.S	151	ρ	\Gr
133	Σ	\GS	152	σ	\Gs
134	•	\ >	153	7	\Gt
135	$\pi$	\pi	154	ω	\Gw
136	д	\.d	155	Δ	\GD
137	≤	\<=	156	Π	\PI
138	$\geq$	\>=	157	Ω	\GW
139	<b>#</b>	\=/	158		\[]
140	α	\Ga	159	00	\00
141	$\rightarrow$	\->	171	«	\<<
142	←	\<-	176	٥	\^0
143	1	\ v	181	μ	\Gm
144	<b>†</b>	\ ^	187	»	\>>
145	γ	\Gg	215	×	\.x
146	δ	\Gd	216	Ø	\0/
189.			247	*	\:-

To avoid any ambiguity during translation and reverse translation:

When data is transferred from the HP 48 with a translate code of 2 or 3, any occurrence of the \ character is replaced by \. For example, R\->B is translated to R\->B. This prevents the reverse translation to R→B when the data is transmitted back to the HP 48.

- When data is transferred to the HP 48 with a translate code of 2 or 3, character sequences beginning with \ are unchanged unless any of the following:
  - They match a sequence in the table.
  - The \ is followed by three decimal digits in the range 000 through 159 for translate code 2.
  - The \ is followed by three decimal digits in the range 000 through 255 for translate code 3.

For example,  $\Gamma Gamma Ga$ 

#### **More About File Names**

In general, the file naming conventions for computers are different than the name requirements for HP 48 variables. When a file is transferred from a computer to the HP 48, the following difficulties may arise due to the computer file name:

- The file name contains characters not allowed in a variable name—for example, AB# or {ABC}. In this case, the HP 48 terminates the transfer and sends an error message to the computer.
- The file name matches a built-in command—for example, SIN or DUP. In this case, the HP 48 appends a number extension to the name—for example, SIN.1.
- The name matches a variable name in the current directory. In this case, to avoid overwriting your variable a number extension is added to the name. (However, if flag -36 is set, the variable will be overwritten.)

Also, an HP 48 file can have a name that is incompatible with the name requirements of the computer software. Transferring such a file can result in a transfer error.

Always check the filenames before a transfer to make sure they are compatible with the receiving system's requirements. If they are not compatible, change the names appropriately.

#### **Errors**

Executing the KERRM command ( NXT KERR ) displays the text of the most recent Kermit error packet.

#### **ASCII and Binary Transmission Modes**

The HP 48 Kermit protocol provides two transfer modes—ASCII and Binary. To get the fastest transfers, you generally should use Binary mode to transfer data from one HP 48 to another, and ASCII mode to transfer data between the HP 48 and a computer.

A receiving HP 48 treats all files as ASCII unless they match the special encoding generated for HP 48 binary files. The calculator will automatically switch to binary receive mode for files with this encoding.

**ASCII Mode.** You *must* use ASCII mode if you want to display, edit, or print your HP 48 file using a computer.

When data is sent from the HP 48 to a computer in ASCII mode:

- The data is converted from its internal HP 48 format to a sequence of characters.
- If the translate code is set to 1, 2, or 3, all line-feed (LF) characters are converted to carriage-return/line feed sequences (CR/LF).
- If the translate code is set to 2 or 3, some or all of the characters with character numbers greater than 127 are translated into displayable character sequences.
- The character sequence \*%HP: modes; is added at the beginning of the data, where modes is a series of characters that describes certain calculator mode settings—the translate, angle, and fraction—mark settings—when the transfer occurred. When this sequence is present, you don't have to set the corresponding modes on the receiving HP 48 when you send the data back.

When data is received by the HP 48 using ASCII mode:

- The data is translated (compiled) into the HP 48 internal format.
- If the translate code is set to 1, 2, or 3, all CR/LFs are converted to LFs.

■ So that the receiving calculator can accurately reconstruct the object being sent by the computer, any modes specified at the beginning of the data are set temporarily in the calculator for the duration of the transfer. If a mode is not specified, the receiving calculator uses its current mode setting.

If you created data (a program, for instance) on your computer, or if you substantially changed data that originally came from your calculator, you may need to include at the beginning of the data the characters \*%HP: modes; where modes is a series of characters—T(), A(), and/or F()—representing the translate code, angle mode, and/or fraction mark. Inside the parentheses are the characters you choose:

- T (translate code) can be followed by 0 (no translation), 1 (translate CR/LF to LF and vice versa), 2 (translate CR/LFs and character numbers 128 through 159), or 3 (translate CR/LFs and character numbers 128 through 255).
- A (angle mode) can be followed by D (degrees), R (radians), or G (grads). If the data contains an angle in degrees, radians, or grads, you should include A(D), A(R), or A(G), respectively.
- F (fraction mark) can be followed by . (period) or , (comma). If it differs from your calculator's setting, the fraction mark used in the data being sent should be included by F(.) or F(,).

For example, at the beginning of the data the sequence %%HP:A(D) will cause the angle mode to be set to degrees during the transfer; %%HP:T(2)A(G)F(,) will cause the translate code to be set to 2, the angle mode to be set to grads, and the fraction mark to be set to comma.

A translate code of T(1) is the normal requirement (and also the system default). You should use T(2) or T(3) only when characters in their respective ranges are being translated according to the table on page 618. You should use T(0) only for string objects, or objects containing string objects, where the string contains binary data.

Binary Mode. In Binary mode, no character conversions are performed. Therefore, the files received from the HP 48 cannot be displayed by the computer. However, if data is being transferred for backup purposes only, Binary mode may be preferable because it is faster, since the data does not require as much processing.

The HP 48 automatically uses Binary mode when transferring libraries, transferring backup objects, or archiving all of user memory.

#### Sending Commands to a Server (PKT)

The PKT command ( NXT PKT ) provides the ability to send and receive data other than HP 48 objects to a remote server. It is particularly useful for sending Kermit commands — for example, Directory (D) or Erase (E).

The PKT command takes two string arguments from the stack—the data field of the packet in level 2, and the packet type in level 1. For example, executing the sequence "D" "G" PKT sends a request for a directory listing.

A server issues one of the following responses to the PKT command:

- An acknowledging message, which is returned to stack level 1.
- An error packet. The HP 48 briefly displays the contents of the error packet. It can be retrieved by executing KERRM (► NXT) KERR.).

#### **Serial Commands**



When using the commands described below to transfer data to or from an HP 48 at 9600 baud, make sure the ticking clock is not in the display. If the clock is in the display, it may interrupt a transfer or corrupt the data

being transferred. The clock display is described on page 439 in chapter 24, "Time, Alarms, and Date Arithmetic."

#### Serial I/O Commands

Keys	Programmable Command	Description
<b>4</b> 1/0 (	page 3):	
XMIT	XMIT	Sends a string in level 1 without Kermit protocol. Once the entire string is sent, a 1 is returned to level 1; if the entire string failed to transmit, a Ø is returned to level 1 and the unsent part of the input string is returned to level 2. Execute ERRM to see the error message.

### Serial I/O Commands (continued)

Keys	Programmable Command	Description
SREEV	SRECV	Receives x characters (argument x is taken from level 1). The characters are returned as a string to level 2, along with a 1 (successful receive) or Ø (unsuccessful receive) to level 1. If the input buffer contains fewer than x characters, the HP 48 will wait the number of seconds specified by the STIME command (the default is 10 seconds). (If the level 2 number returned by the BUFLEN command (see BUFLE below) is used as the argument for SRECV, no waiting will occur because x will exactly match the number of characters in the input buffer.) In the event of an unsuccessful receive, executing ERRM returns the error message associated with the failure.
STIME	STIME	Sets the serial transmit/receive timeout to x seconds (argument x is taken from level 1). The value for x can range from 0 to 25.4 seconds. If 0 is used, no timeout will occur (which could result in excessive battery drain).
SBRK	SBRK	Sends a serial BREAK.

#### Serial I/O Commands (continued)

Keys	Programmable Command	Description
BUFLE	BUFLEN	Returns the number of characters in the HP 48 input buffer to level 2, along with a 1 (no framing error or UART overrun) or a Ø (framing error or UART overrun) to level 1. If a Ø is returned, the number of characters returned to level 2 represents the part of the data received before the error. Therefore, that number can be used to determine where the error occurred.



Even though XMIT, SRECV, and BUFLEN check the send and receive mechanisms, the integrity of the data is not checked. One method to insure that the data sent is the same as the data received involves appending a checksum

to the end of the data being sent, and then verifying that checksum at the receiving end.

XMIT, SRECV, and SBRK automatically open the IR/serial port using the current values of the first four IOPAR parameters (baud, parity, receive pacing, and transmit pacing) and the current IR/wire setting (set using IRZW in the I/O SETUP menu).

# **Using Plug-in Cards and Libraries**



#### This chapter covers:

- The types of memory and plug-in cards.
- Installing and removing plug-in cards.
- Using RAM cards to expand user memory or to back up data.
- Using application cards and libraries.

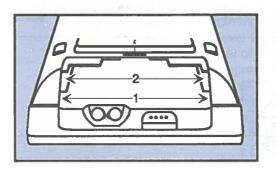
# **Types of Memory**

Plug-in cards increase the amount of HP 48 memory. The HP 48 has two types of memory:

- Read-only memory, or ROM, is memory that cannot be altered. The HP 48 has 256K bytes of built-in ROM that contains its command set. You can expand the amount of ROM by installing plug-in application cards.
- Random-access memory, or RAM, is memory you can change. You can store data into RAM, modify its contents, and purge data. The HP 48 contains 32K bytes of built-in RAM. You can increase the amount of RAM by adding plug-in RAM cards.

# **Installing and Removing Plug-In Cards**

The HP 48 has two *ports* for installing plug-in cards, designated port 1 and port 2. Port 1 is closest to the front of the calculator; port 2 is closest to the back. Cards can be installed in either port.





The calculator must be turned off while you are installing or removing plug-in cards. Otherwise, all of user memory could be erased.

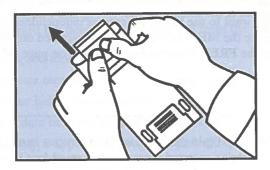
Also, whenever a card is installed or removed, the HP 48 executes a system halt, causing the contents of the stack to be lost.

#### To install a plug-in card:

- 1. If you are installing a new RAM card, first install its battery (see "Installing the Battery in a New RAM Card," page 639) and set the write-protect switch to the desired position (see "Setting the Write-Protect Switch" on page 641).
- 2. Turn off the calculator. Do not press ON until you've completed the installation procedures.

3. Remove the port cover at the top of the calculator by pressing down against the grip area and then pushing in the direction shown.

Removing the cover exposes the two plug-in ports.



- 4. Select an empty port for the card—either port may be used.
- 5. Position the plug-in card as shown. The triangular arrow on the card must point down, toward the calculator. Make sure the card is lined up properly with a port opening and not positioned half in one port and half in the other.



- 6. Slide the card firmly into the port until it stops. When you first feel resistance, the card has about 1/4" to go to be fully seated.
- 7. If desired, repeat steps 4 through 6 for another card.
- 8. Replace the port cover by sliding it on until the latch engages.

- 9. If the card is a RAM card, you must decide how you want to use it (see page 642):
  - If you want to use the RAM card to increase user memory, execute the MERGE command as described on page 643.
  - If you want to use the RAM card as independent memory, execute the MERGE command as described on page 643 and then the FREE command as described on page 649.

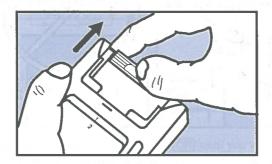
#### To remove a plug-in card:



If the plug-in card you want to remove is a RAM card that contains merged memory, you must free the merged memory before removal. Failure to do so would probably result in loss of data stored in user memory.

See "Freeing Merged Memory" on page 649 for instructions.

- **1.** Turn off the calculator. Do not press ON until you've completed the removal process.
- 2. Remove the port cover.
- 3. To remove a card, press against the grip as shown and slide the card out of the port.



4. Replace the port cover.

#### **RAM Cards**

RAM cards let you increase the amount of RAM in your HP 48. Each card contains a battery that preserves its contents when the calculator is off or when the card has been properly removed from the calculator.

#### RAM cards are good tools for:

- Expanding user memory.
- Backing up or hiding important data.
- Exchanging data between two HP 48 calculators.
- Storing prototype application programs that will eventually be made into ROMs.

#### **Preparing the Card for Installation**

Installing the Battery in a New RAM Card. Before a new RAM card is installed, the battery that came with it must be installed in the card.

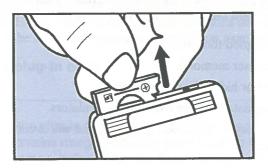


Do not use this procedure for *replacing* a battery in a RAM card—it could cause loss of memory in the RAM card. Appendix A contains instructions for replacing RAM card batteries on page 663.

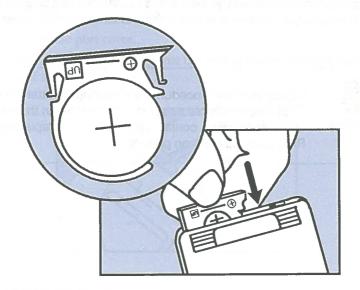
<sup>&</sup>quot;Uses for RAM Cards" on page 642 covers these tasks.

To install the battery in a new RAM card:

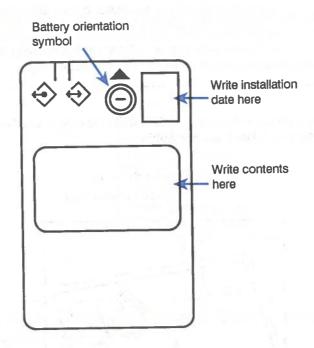
1. Remove the battery holder from the card by inserting a thumbnail or small screwdriver into the groove and pulling in the direction shown.



2. The grooved side of the battery holder is marked with the + symbol and the word UP. Insert the battery into the holder with its + side up, and then slide the holder into the card.



3. Write the date of installation on the card using a fine-point, permanent marker. The date is important for determining when to replace the battery.



4. Set an alarm in the calculator for 1 year from the date of installation to remind you to replace the battery. (Depending on the use, the battery should last between 1 and 3 years. When the battery needs replacing, a display message will appear if the card is in the calculator. You are setting this alarm in case the card is not in the calculator when the battery gets low.) Setting alarms is covered in chapter 24, and replacing RAM-card batteries is covered in appendix A.

Setting the Write-Protect Switch. The write-protect switch lets you protect the contents of the RAM card from being accidentally overwritten or erased. The switch has two positions:

- Read-only. The contents of the RAM card can be read, but cannot be changed or erased.
- Read/write. You can write information to the RAM card and erase its contents.

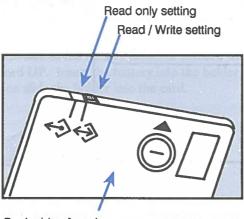


#### Caution

To avoid loss of user memory:

- Always turn off the calculator before changing the write-protect switch on an installed card.
- Do not write protect a RAM card containing merged memory; the memory should be freed first (see page 649).

You can operate the write-protect switch while the card is installed; however, the switch labels are not visible.



Back side of card

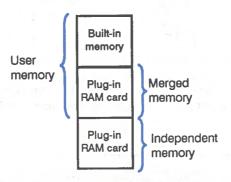
#### **Uses for RAM Cards**

A RAM card can be used in one of two ways:

- It can be *merged* with built-in memory. This enables you to expand the amount of user memory available (up to 288K bytes) for creating variables and directories, putting objects on the stack, etc.
- It can provide a place *independent* of user memory in which to back up important data. You can copy individual objects or entire directories to a RAM card in much the same way as you would back up computer files to a disk. After you've copied the data, you can remove the card and store it in a safe place, or, as a way of transferring data, install the card in another HP 48.

You can install one or two RAM cards, and you can use either or both of them for either purpose. However, you cannot use a single card for both merged and independent memory at the same time.

The following diagram illustrates a system containing two RAM cards—one containing merged memory and the other containing independent memory.

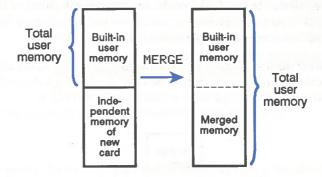


# Using RAM Cards to Expand User Memory (Merged Memory)

Before you can use an installed RAM card to expand user memory, you must execute the MERGE command to *merge* its memory with built-in memory.

Before you execute the MERGE command, the write-protect switch on the RAM card must be in the read/write position. (See page 641 for how to set the write-protect switch.)

MERGE takes a port number as its argument. For example, the keystrokes 1 — MEMORY NXT NXT MERG merge the plug-in memory installed in port 1 with built-in memory.



When you merge a RAM card that contains backup objects, those objects are moved to a special port, called port 0. (See page 647 for a description of port 0.)



You should never remove a RAM card that contains merged memory. Doing so will cause loss of data stored in user memory. Before you can remove the RAM card, you must *free* the merged memory. (See "Freeing

Merged Memory" on page 649 for instructions.) If you accidentally remove a card with merged memory and see the message Replace RAM, Press ON, you can minimize memory loss by leaving the calculator on, reinserting the card in the same port, and then pressing ON.

# Using RAM Cards for Backup (Independent Memory)

The HP 48 uses a special object type, the backup object, to store backedup data. A backup object contains another object, its name, and its checksum. Simply put, a backup object contains a variable or directory and its checksum. An independent-memory RAM card that contains the backup objects can be removed from the HP 48 and either stored for later use or transferred to another HP 48.

### **Backing Up Objects into Independent Memory**

Backup objects can exist:

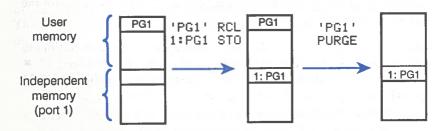
- In independent memory (port 1 and/or port 2).
- In a portion of user memory called port 0 (see page 647).

To create a backup object, execute the STO command with two arguments—the object to be backed up in level 2, and a backup identifier in level 1. A backup identifier has this form:

#### :port#:name

where port# is the port number (0, 1, or 2) and name is the name under which the backup copy will be stored.

**Example: Backing Up a Program.** To back up a program named *PG1* into independent memory in port 1, recall the program to the stack by evaluating the sequence 'PG1' RCL, and then store the object as a backup object in port 1 by evaluating :1:PG1 STO.



The backup object in the previous example happens to have the same name as the original object, but the two names could be different.

Note that a directory and its subdirectories can be backed up in a single backup object.

Example: Backing Up a Directory and Its Subdirectories.

Suppose your HOME directory contains a subdirectory named CHEM, which in turn contains several subdirectories. To back up the entire directory structure of CHEM in a backup object named BCHEM, recall the directory to the stack by evaluating the sequence 'CHEM' RCL, and then store it in the backup object by evaluating :1:BCHEM STO.

#### **Accessing Backup Objects**

You can recall, evaluate, and purge the contents of backup objects. You can also obtain a listing of all the backup objects in a given port.

Recalling Backup Objects. The LIBRARY menu can be used to recall the contents of backup objects. Pressing LIBRARY followed by PORTO, PORTO displays a menu of backup objects and libraries in that port. To recall the contents of a backup object to the stack, simply press and then the menu key for the desired backup object.

The RCL command can also be used to recall the contents of a backup object to the stack. For example, evaluating the sequence 11:BPG1 RCL recalls the object stored in 1:BPG1.

Evaluating Backup Objects. To use the LIBRARY menu to evaluate the contents of a backup object, press [LIBRARY] followed by PORTO, PORTO, or PORTO. Then, simply press the menu key for the desired backup object.

Also, when the argument of EVAL is a backup name, the contents of the backup object is evaluated. For example, executing the sequence :1:BPG1 EVAL evaluates the program stored in backup object 1:BPG1. (EVAL also takes a list of backup objects as its argument to evaluate more than one at a time.)

Purging Backup Objects. To purge a backup object, use the backup name as the argument of PURGE. For example, executing the sequence :1:BPG1 PURGE purges the backup object. (PURGE can take a list of backup objects as its argument to purge more than one at a time.)

Using Wildcards to RCL, EVAL, and PURGE. The character & can be used as a wildcard to replace the port number in the arguments used by RCL, EVAL, and PURGE. (& is the left-shifted alpha key above ENTER.) When the HP 48 encounters the wildcard with these commands, it searches port 2, 1, 0, and then main memory for the accompanying backup object (the first occurrence of the name is used). For example, evaluating the sequence : &: BPG1 PURGE causes the HP 48 to search port 2, 1, 0, and then main memory for the first occurrence of BPG1 and then delete it.

NXT PYRRS) can be used to display a list of objects in the specified port. It takes as its argument a port number 0, 1, or 2. It returns to level 1 the type of memory contained in the port ("ROM", "SYSRAM", or a number representing the amount of free independent RAM); and to level 2 it returns a list of backup objects and library identification numbers (both tagged with the port number).

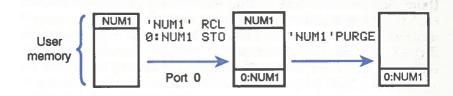
Also, you can use the LIBRARY menu to display a menu of backup objects and libraries in a given port. Simply press LIBRARY followed by PORTO, PORTO, PORTO to see the desired menu.

# **Backing Up Objects into User Memory (Port 0)**

The HP 48 lets you create backup objects in user memory. The portion of user memory used for backup objects and libraries is called "port 0." There are several reasons you might want to back up data into user memory:

- You want to "hide" data; that is, you want certain data to be in user memory, but you don't want the variable(s) to appear in any directory.
- You want to "free" a RAM card being used for merged memory, and instead use it for independent memory. (See "Freeing Merged Memory" on page 649).

You create a backup object in user memory the same way you create other backup objects, except you specify port 0 as the port number.



The size of port 0 is dynamic—it grows and shrinks to accommodate its contents.

#### **Backing Up All of Memory**

The ARCHIVE command ( MEMORY NXT NXT ARCHI) creates a backup object named : port#: name in independent memory containing a copy of:

- The entire HOME directory.
- User key assignments.
- The alarm catalog.

It takes a name tagged by a port number (0, 1, or 2) as its argument. For example, executing the sequence :2:JUN12 ARCHIVE creates backup object :2:JUN12.

The RESTORE command ( NXT) NXT) RESTO retrieves the data backed up by the ARCHIVE command. It, too, takes a name (where the corresponding object is a directory) tagged by a port number as its argument. For example, executing the sequence 2: JUN12 RESTORE retrieves the HOME directory backed up above.



Executing RESTORE *overwrites* the entire contents of user memory with the contents of the backup object.

If you want your flag settings to be saved when you back up all of memory, recall them to the stack (using RCLF) and store them in a variable before executing ARCHIVE. After you RESTORE memory, you can reactivate the flag settings by recalling the contents of that variable to the stack and executing STOF (store flags).

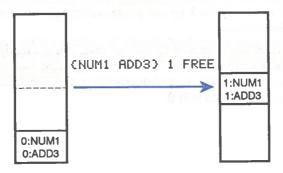
# Freeing Merged Memory

Freeing merged memory converts it to independent memory. Merged memory must be freed if:

- You want to remove the RAM card from its port.
- You want to use the RAM card as independent memory, rather than user memory.

The FREE command (MEMORY NXT NXT FREE) frees the merged memory in a specified port. It takes two arguments—a list in level 2, and the port number in level 1.

The list can be empty, in which case the merged memory is simply freed, or it can contain one or more names or library identifiers. If the list is not empty, FREE moves the named backup objects and libraries from port 0 into the newly-freed card. For example, executing the sequence { NUM1 ADD3 } 1 FREE frees the merged memory in port 1 and makes it independent memory. At the same time, the backup objects NUM1 and ADD3 in port 0 are moved to port 1.



To free merged memory, first execute MEM to determine the amount of available memory (press MEMORY MEM). If the amount of available memory is greater than or equal to the amount of memory on the card you are going to free, you are ready to execute the FREE command.

If MEM returns a value less than the amount of memory on the card, executing FREE without any preparation would return an error, since your stored data would not fit into the amount of user memory remaining after the merged memory was freed. To avoid an error, you can do any of the following:

- Purge unneeded variables from user memory.
- Back up data into another RAM card installed in the other port and then purge the original variables.
- Back up data into port 0 (built-in memory) and then use the level-2 argument of the FREE command to move that data into the freed memory. Here's a step-wise procedure for doing this:
  - 1. Determine the amount of data that must be moved into the memory that you'll be freeing. For example, if you'll be removing a 128K RAM card, and the amount of user memory available is 126K, you must move at least 2K of variables.
  - 2. Back up the variable in port 0. For example, to back up CALC1 into port 0, recall its contents to the stack and execute : 0: CALC1 STO.
  - 3. Purge the variable from user memory (for example, 'CALC' PURGE).
  - **4.** If necessary, back up and purge additional variables and directories.
  - 5. When you've backed up enough data, you are ready to execute the FREE command. The level-2 argument must be a list containing the names of the variables and directories you've backed up into port 0.

# **Using Application Cards and Libraries**

A library is an object that contains named objects that can act as an extension to the built-in command set. You cannot view or change the contents of a library. Libraries can exist in application cards, or they may be copied into RAM. However, libraries cannot be created by the HP 48.

#### Libraries are identified by:

- A library identifier, which has the form : port#: library#. The library# (library number) is a unique number associated with the library. The library identifier is used as the argument of commands that work with library objects.
- The *library name*, which is a sequence of characters. The library name appears in the LIBRARY menu when the library is attached to a directory on the current path.

#### Attaching a Library to a Directory

To use a library, it must be attached to a directory in user memory. The attachment may happen automatically when you install an application card, or you may have to do it yourself. Consult the owner's documentation accompanying your application card (or RAM-based library) for information about attaching the library.

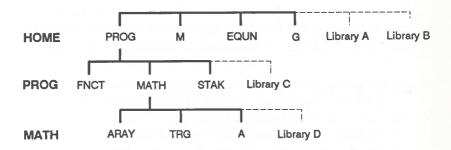
If the library is not attached automatically, you must use the ATTACH command (MEMORY NXT) HITHC) to attach it. ATTACH requires a library number as its argument.

This is no limit on the number of libraries that can be attached to the HOME directory. Only one library at a time can be attached to a particular subdirectory.

#### **Accessing Library Operations (The LIBRARY Menu)**

The LIBRARY Menu. Pressing displays the LIBRARY menu, which contains the names of the libraries on the current directory path. To display a menu of the operations in a library, press the appropriate key. For example, if you have the HP Solve Equation Library installed in your calculator, pressing LIBRARY EQUIP displays a menu of all the operations in that library.

Accessing Libraries Attached to Subdirectories. The rules for accessing libraries attached to various subdirectories are the same as the rules for accessing variables in those directories. For example, suppose your HP 48 has the following directory structure and attached libraries:



Like variables, library operations can be accessed if the library is attached to the current directory or to a directory in the current path. For example, since libraries A and B are attached to *HOME*, their operations can be accessed from any directory. You can access the operations in library C when PROG or MATH are the current directory. However, you cannot access the operations in library D when PROG is the current directory.

## **Additional Commands That Access Libraries**

#### **Library Commands**

Keys	Programmable Command	Description
STO	STO	Stores a library object from level 2 into independent memory in the port specified in level 1.
r→ RCL	RCL	Takes a library identifier (: port#: library#) as its argument and recalls the specified library to the stack.
PURGE	PURGE	Takes a library identifier (: port#: library#) as its argument and purges the specified RAM-based library.
MEMORY (page 2):		
PVARS	PVARS	Takes a port number as its argument and displays a list of the backup identifiers and library identifiers in the specified port.
LIBS	LIBS	Displays a list containing the names, library number, and port number of all the libraries attached to the current directory.
ATTAC	ATTACH	Takes a library number as its argument and attaches the specified library to the current directory.
DETAC	DETACH	Takes a library number as its argument and detaches the specified library from the current directory.

# **Appendixes and Indexes**

# Support, Batteries, and Service

## **Calculator Support**

You can obtain answers to questions about using your calculator from our Calculator Support department. Our experience has shown that many customers have similar questions about our products, so we have provided the following section, "Answers to Common Questions." If you don't find the answer to your question there, contact us at the address or phone number on the inside back cover.

#### **Answers to Common Questions**

**Q:** The calculator doesn't turn on when I press ON. What's wrong?

**A:** There may be a simple problem that you can solve immediately, or the calculator may require service. See "Testing Calculator Operation" on page 665.

**Q:** I'm not sure whether the calculator is malfunctioning or if I'm doing something incorrectly. How can I verify that the calculator is operating properly?

A: Refer to "Testing Calculator Operation" on page 665 in this appendix.

**Q:** The (•) annunciator stays on even when the calculator is turned off. Is anything wrong?

A: This indicates a low-battery condition in the calculator or a RAM card, or an alarm that is past due. To determine what is causing the (\*\*) annunciator to stay on, turn the calculator off and then on. A message in the display will identify the problem. Refer to "Changing Batteries" in this appendix (page 661) or to "Setting Alarms" in chapter 24 (page 443).

**Q:** How can I determine how much memory is left in the calculator?

A: Press MEMORY MEM. The number of bytes of available memory will appear at the lower right corner of the display. An empty memory should show approximately 30000 (bytes of internal RAM).

Q: How do I clear everything from the calculator's memory?

A: Perform the following steps:

- 1. Press and hold ON.
- 2. Simultaneously press and release both of the outer keys in the top row (the menu keys with A and F next to them).
- 3. Release ON.

The calculator will beep and the Try To Recover Memory? prompt will be displayed. Press NO to clear user memory; the Memory Clear message will appear in the display.



This procedure will not clear the contents of a plug-in RAM card unless that RAM is merged with the calculator's main memory.

Q: How do I change the number of decimal places the HP 48 displays?

A: Perform the following steps:

- 1. Go to page 1 of the MODES menu: press MODES.
- **2.** Press the number of decimal places you want (0 11).
- 3. Press the menu key for the display format you desire (FIX, SCI, or ENG).

Refer to "Display Modes" in chapter 2 (page 57).

**Q:** My numbers contain commas as decimal points. How do I restore periods?

A: Perform the following steps:

- 1. Go to page 4 of the MODES menu (press MODES NXT NXT).
- 2. Press the FM₁ radix toggle menu key. (The should disappear from the menu key.)

Q: What does an "E" in a number (for example, 2.51E-13) mean?

A: Exponent of 10 (for example,  $2.51 \times 10^{-13}$ ). Refer to "Display Modes" (page 57) in chapter 2.

**Q:** When I take the sine of  $\pi$  in Degrees mode, why do I get 'SIN( $\pi$ )' instead of a number?

A: The calculator is in Symbolic Result mode;  $|SIN(\pi)|$  is the symbolic answer. Press  $\rightarrow NUM$  to convert  $|SIN(\pi)|$  to its numerical equivalent of .0548... up to 11 decimal places. You can also press |SYM| on page 1 of the MODES menu to change to Numerical Results mode and prevent symbolic evaluation.

Q: What does "object" mean?

A: "Object" is the general term for all elements of data the HP 48 works with. Numbers, expressions, arrays, programs, and so on, are all types of objects. Refer to chapter 4, "Objects," for a description of the object types accepted by the calculator.

Q: What do three dots (...) mean at either end of a display line?

A: The three dots (called an *ellipsis*) indicate that the displayed object is too long to display on one line. To view undisplayed portions of the object, use the operation of the object, use the operation of the object, use the operation of t

Q: The calculator beeps and displays Bad Argument Type. What's wrong?

A: The objects on the stack aren't the correct type for the command you are attempting. For example, executing \*\*UNIT\* (in page 2 of the PRG OBJ menu) with a number in stack levels 1 and 2 causes this error.

Q: The calculator beeps and displays Too Few Arguments. What's wrong?

A: There are fewer arguments on the stack than required by the command you are attempting. For example, executing + with only one argument or number on the stack causes this error.

**Q:** The calculator beeps and displays a message different from the two listed above. How do I find out what's wrong?

A: Refer to "Messages" in appendix B.

Q: I can't find some variables that I used earlier. Where did they go?

A: You may have been using the variables in a different directory. If you can't remember which directory you were using, you'll need to check all the directories in your calculator.

Q: Sometimes my HP 48 seems to pause for a few seconds during a calculation. Is anything wrong?

A: Nothing is wrong. The calculator does some system cleanup from time to time to eliminate temporary objects created from normal operation. This cleanup process frees memory for current operations.

**Q:** During normal operation, the printer prints several lines quickly, then slows down. Why?

A: The calculator quickly transmits a certain amount of data to the printer, then slows its transmission rate to ensure that the printer can keep up.

**Q:** How can I increase the printing speed of my HP 82240B Infrared Thermal Printer?

A: Use an agc adapter with your HP 82240B printer so that the printer can print faster. Also, set the calculator delay to match the print speed (see "Setting the Delay" on page 607).

#### **Environmental Limits**

Calculator. To maintain product reliability, avoid getting the calculator wet and observe the following temperature and humidity limits:

- Operating temperature: 0° to 45°C (32° to 113°F).
- Storage temperature: -20° to 65°C (-4° to 149°F).
- Operating and storage humidity: 90% relative humidity at 40°C (104°F) maximum.

Plug-In Cards. The environmental limits for Hewlett Packard plug-in cards are:

- Operating temperature: 0° to 45°C (32° to 113°F).
- Storage temperature: -20° to 60°C (-4° to 140°F).
- Storage temperature for RAM card data retention: 0° to 60°C (32° to 140°F).
- Operating and storage humidity: 90% relative humidity at 40°C (104°F) maximum.

# When to Replace Batteries

When a low-battery condition exists, the (\*) annunciator remains on, even when the calculator is turned off. When the calculator is turned on during a low-battery condition, Warning: LowBat() is displayed for approximately 3 seconds. LowBat(P1) refers to port 1, LowBat(P2) refers to port 2, and LowBat(S) refers to the calculator (system) batteries.

Replace the RAM card battery or the calculator batteries as soon as possible after the (\*\*) low-battery annunciator and warning message appear. If you continue to use the calculator while the (\*\*) annunciator is on, the display will eventually dim and you may lose calculator and RAM card data.

Under typical use, a RAM card's battery should last between 1 and 3 years. Be sure to mark the card with the battery-installation date, and, in case the RAM card is not in the calculator when the battery needs replacement, set an alarm for 1 year from that date to remind you to install a fresh battery. RAM cards do not come with a battery installed.

## **Changing Batteries**

#### **Battery Types**

**Calculator Batteries.** Any brand of size AAA batteries. Be sure that all three batteries are of the same brand and type.

The use of rechargeable batteries is not recommended because of their lower capacity.

Plug-In RAM Card Batteries. 3-Volt 2016 coin cell.

#### **Changing Calculator Batteries**

These instructions are for changing calculator batteries. The instructions for replacing RAM card batteries start on page 663.



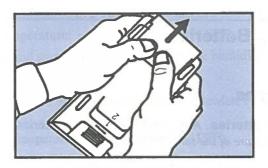
Caution

Whenever you remove batteries from the calculator, be sure the calculator is off and do not press the ON key until the new batteries are installed. If you press ON when batteries are not in the calculator, you may lose all

of calculator memory.

1. Turn the calculator off. You may lose memory in the calculator and plug-in RAM cards if the calculator batteries are removed when the calculator is on.

- 2. Have three, fresh batteries (of the same brand and type) at hand. Wipe off both ends of each battery with a clean, dry cloth.
- 3. Remove the calculator battery-compartment door by pressing down and sliding it off away from the calculator. Be careful not to press the calculator's ON key. Refer to the following illustration:



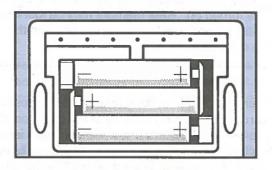
4. Turn the calculator over and shake the batteries out. Once the batteries are out, you should replace them with fresh batteries within 2 minutes to protect against memory loss.



Do not mutilate, puncture, or dispose of batteries in fire. The batteries can burst or explode, releasing hazardous chemicals. Discard used batteries according to the manufacturer's instructions.

5. Avoid touching the battery terminals. Batteries are easier to install if the negative (plain) ends are inserted first, and if the center battery is installed last.

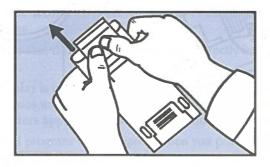
Position the batteries according to the outlines in the bottom of the battery compartment. Refer to the following illustration:



- **6.** Replace the battery-compartment door by sliding the tabs on the door into the slots in the calculator case.
- 7. Press ON to turn the calculator on.

#### **Changing a RAM Card Battery**

1. Turn the calculator over and remove the plastic cover over the plug-in card ports (on the display-end of the calculator).



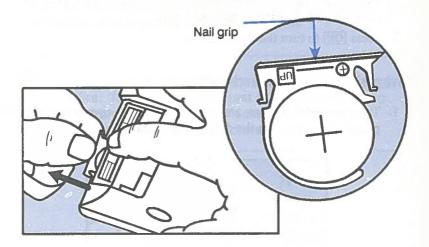
2. With the RAM card in port 1 or 2, turn the calculator on.



Since RAM cards run off the calculator batteries when the calculator is ON, you should replace a card's battery only when the card is in the calculator and the calculator is turned on. RAM memory may be lost if you remove a

RAM card battery when the calculator is off, or when the card is not installed in the calculator.

3. Place your index finger in the recess near the exposed end of the RAM card—this prevents removal of the card from the calculator when you remove the card's battery holder. Now insert the thumbnail of your free hand into the nail grip in the black plastic at the left side of the end of the card and pull the battery holder out of the card.



4. Remove the old battery from the plastic battery holder.



Do not mutilate, puncture, or dispose of batteries in fire. The batteries can burst or explode, releasing hazardous chemicals. Discard used batteries according to the manufacturer's instructions.

- 5. Install a fresh, 3-Volt 2016 coin cell in the plastic battery holder and reinsert the battery holder (with battery) into the RAM card. Be sure to install the battery with the side marked "+" toward the front of the card.
- 6. Mark the card with the battery-installation date, and, in case the RAM card is not in the calculator when its battery needs replacing, set an alarm for 1 year from that date to remind you to change it.
- 7. Replace the plug-in port cover.

# **Testing Calculator Operation**

Use the following guidelines to determine whether the calculator is functioning properly. Test the calculator after every step to see if operation has been restored. If your calculator requires service, refer to page 674.

# The calculator won't turn on or doesn't respond when you press the keys.

- 1. Make sure that three fresh batteries are correctly installed in the calculator.
- 2. If the display is blank, press and hold ON; press and release + several times until characters become visible; then release ON. If no characters appear in the display, the calculator requires service.
- 3. If a halted program won't respond when you press ATTN, try pressing ATTN again.

- 4. If the keyboard is "locked," perform a system halt as follows:
  - a. Press and hold ON.
  - **b.** Press and release the third key from the left in the top row (the menu key with C next to it).
  - c. Release ON.

The empty stack display should appear.

- 5. If the display appears garbled, perform a memory reset as follows:
  - a. Press and hold ON.
  - **b.** Press and hold both of the outer keys in the top row (the menu keys with A and F next to them).
  - c. Release all three keys.

The calculator will beep and display the message Try To Recover Memory? at the top of the display. Press YES to recover as much memory as possible.

If these steps fail to restore operation, the calculator requires service.

# The calculator responds to keystrokes, but you suspect it's malfunctioning.

- 1. Run the self-test described in the next section. If the calculator fails the self-test, it requires service.
- 2. If the calculator passes the self-test, you may have made a mistake operating the calculator. Reread appropriate portions of the manual and check "Answers to Common Questions" (page 656).
- **3.** Contact the Calculator Support department. The address and phone number are listed on the inside back cover.

#### **Self-Test**

If the display turns on, but the calculator does not seem to be operating properly, run the diagnostic self-test:

- 1. Press and hold ON.
- 2. Press and release the second key from the right in the top row (the menu key with E next to it).
- 3. Release ON.

The diagnostic self-test tests the internal ROM and RAM, and generates various patterns in the display. The test repeats continuously until it is halted.

- 4. To halt the self-test, perform a system halt as follows:
  - a. Press and hold ON.
  - **b.** Press and release the third key from the left in the top row (the menu key with C next to it).
  - c. Release ON.

The empty stack display should appear.

The diagnostic self-test should be successfully completed before running any of the tests described in the following sections.

If the self-test indicates an internal ROM or RAM failure (if IROM OK and IRAM OK are not displayed), the calculator requires service.

# **Keyboard Test**

This test checks all of the calculator's keys for proper operation.

To run the interactive keyboard test:

- 1. Press and hold ON.
- 2. Press and release the third key from the right in the top row (the menu key with D next to it).
- 3. Release ON.

- 4. Press and release the second key from the right in the top row (the menu key with E next to it). KBD1 will appear in the upper left corner of the display.
- 5. Starting at the upper left corner and moving left to right, press each of the 49 keys on the keyboard. If you press the keys in the proper order and they are functioning properly, the calculator emits a high-pitch beep at each press of a key. When the 49th key ( + ) has been pressed, the displayed message should change to KBD1 OK.

If you press a key out of sequence, a five-digit hexadecimal number will appear next to KBD1. Reset the keyboard test (do steps 1 through 3 above), and rerun the test.

If a key isn't functioning properly, the next keystroke displays the hex location of the expected and the received location. If you pressed the keys in order and got this message, the calculator requires service. Be sure to include a copy of the error message when you ship the calculator for service.

- 6. To exit the keyboard test, perform a system halt as follows:
  - a. Press and hold ON.
  - **b.** Press and release the third key from the left in the top row (the menu key with C next to it).
  - c. Release ON.

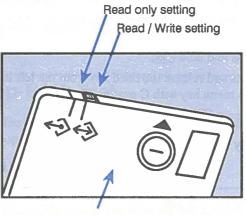
The empty stack display should appear.

#### **Port RAM Test**

The port RAM test non-destructively tests the ports and the installed plug-in RAM cards. (Plug-in RAM-card memory is preserved.)

#### To run the port RAM test:

- 1. Check that a plug-in RAM card is properly installed in port 1 and/or port 2.
- 2. Verify that the switch on each card is set to the "read/write" position:



Back side of card

- 3. Turn the calculator on.
- 4. Press and hold ON.
- 5. Press and release the fourth key from the left in the top row (the menu key with D next to it).
- 6. Release ON.

A vertical line will appear at both sides and at the center of the display.

#### 7. Press and release .

RAM1 and/or RAM2 will appear at the top left corner of the display and the size of the corresponding plug-in RAM card (32K or 128K) will appear at the top right corner of the display. OK will appear to the right of RAM1 and/or RAM2 when the port RAM test has been successfully completed. A failure message (for example, RAM1 00002) will be displayed for each port that does not contain a plug-in RAM card or if a card's read/write switch is in the "write-protect" position. This message should be ignored.

If OK does not appear for a RAM card set to read/write, the card should be moved to the other port and the test rerun. IF OK still doesn't appear, the RAM card should be replaced with a new one.

- 8. To return to normal calculator operation, perform a system halt as follows:
  - a. Press and hold ON.
  - **b.** Press and release the third key from the left in the top row (the menu key with C next to it).
  - c. Release ON.

The empty stack display should appear.

## **IR Loop-Back Test**

This test checks the operation of the send and receive infrared sensors and their associated circuits.

To run the IR Loop-Back test:

- 1. Press and hold ON.
- 2. Press and release the fourth key from the left in the top row (the menu key with D next to it).
- 3. Release ON; a vertical line will appear at both sides, and at the center of the display.
- 4. Be sure that the plastic plug-in card cover is in place and that it covers the clear lamp bulbs in the top end of the calculator.

#### 5. Press EVAL.

IRLB will appear at the top left corner of the display.

OK will appear to the right of IRLB if the calculator passes this test.

If OK does not appear, the calculator requires service.

- **6.** To return to normal calculator operation, perform a system halt as follows:
  - a. Press and hold ON.
  - **b.** Press and release the third key from the left in the top row (the menu key with C next to it).
  - c. Release ON.

The empty stack display should appear.

# **Serial Loop-Back Test**

This test checks the operation of the send and receive circuits of the serial interface at the top of the calculator.

To run the Serial Loop-Back test:

- 1. Press and hold ON.
- 2. Press and release the fourth key from the left in the top row (the menu key with D next to it).
- 3. Release ON; a vertical line will appear at both sides, and at the center of the display.
- 4. Temporarily connect (short) the middle two pins (pins 2 and 3) of the 4-pin serial connector at the top end of the calculator. Be careful not to bend or severely jar the pins.

#### 5. Press PRG.

U\_LB will appear at the top left corner of the display.

OK will appear to the right of U\_LB if the calculator passes this test.

If OK does not appear, the calculator requires service.



If you inadvertently short pins 1 and 2 or pins 3 and 4 of the serial connector, the loop-back test will return U\_LB 00001 or U\_LB 00002 (test-failed message), but you will not damage the calculator.

- **6.** To return to normal calculator operation, perform a system halt as follows:
  - a. Press and hold ON.
  - **b.** Press and release the third key from the left in the top row (the menu key with C next to it).
  - c. Release ON.

The empty stack display should appear.

# **Limited One-Year Warranty**

What Is Covered. The calculator (except for the batteries, or damage caused by the batteries) and calculator accessories are warranted by Hewlett-Packard against defects in materials and workmanship for one year from the date of original purchase. If you sell your unit or give it as a gift, the warranty is automatically transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center. (Replacement may be made with a newer model of equal or better functionality.)

This warranty gives you specific legal rights, and you may also have other rights that vary from state to state, province to province, or country to country.

What Is Not Covered. Batteries, and damage caused by the batteries, are not covered by the Hewlett-Packard warranty. Check with the battery manufacturer about battery and battery leakage warranties.

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY. Some states, provinces, or countries do not allow

limitations on how long an implied warranty lasts, so the above limitation may not apply to you. IN NO EVENT SHALL HEWLETT-PACKARD COMPANY BE LIABLE FOR CONSEQUENTIAL DAMAGES.

Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products, once sold.

Consumer Transactions in the United Kingdom. This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

## If the Calculator Requires Service



If the contents of your calculator's memory are important, you should back up the memory on a plug-in RAM card, another HP 48, or a computer before sending in the calculator for repair.

Hewlett-Packard maintains service centers in many countries. These centers will repair a calculator, or replace it with the same model or one of equal or better functionality, whether it is under warranty or not. There is a service charge for service after the warranty period. Calculators normally are serviced and reshipped within 5 working days.

- In the United States: Send the calculator to the Corvallis Service Center listed on the inside of the back cover.
- In Europe: Contact your Hewlett-Packard sales office or dealer, or Hewlett-Packard's European headquarters (address below) for the location of the nearest service center. Do not ship the calculator for service without first contacting a Hewlett-Packard office.

Hewlett-Packard S.A. 150, Route du Nant-d'Avril P.O. Box CH 1217 Meyrin 2 Geneva, Switzerland Telephone: 022 780.81.11

■ In other countries: Contact your Hewlett-Packard sales office or dealer or write to the Corvallis Service Center (listed on the inside of the back cover) for the location of other service centers. If local service is unavailable, you can ship the calculator to the Corvallis Service Center for repair.

All shipping, reimportation arrangements, and customs costs are your responsibility.

**Service Charge.** Contact the Corvallis Service Center (inside back cover) for the standard out-of-warranty repair charges. This charge is subject to the customer's local sales or value-added tax wherever applicable.

Calculator products damaged by accident or misuse are not covered by the fixed charges. These charges are individually determined based on time and material.

**Shipping Instructions.** If your calculator requires service, ship it to the nearest authorized service center or collection point.

- Include your return address and a description of the problem.
- Include proof of purchase date if the warranty has not expired.
- Include a purchase order, check, or credit card number plus expiration date (VISA or MasterCard) to cover the standard repair charge.
- Ship your calculator postage *prepaid* in adequate protective packaging to prevent damage. Shipping damage is not covered by the warranty, so we recommend that you insure the shipment.

**Warranty on Service.** Service is warranted against defects in materials and workmanship for 90 days from the date of service.

**Service Agreements.** In the U.S., a support agreement is available for repair and service. For additional information, contact the Corvallis Service Center (see the inside of the back cover).

## **Regulatory Information**

**U.S.A.** The HP 48 generates and uses radio frequency energy and may interfere with radio and television reception. The calculator complies with the limits for a Class B computing device as specified in Subpart J of Part 15 of FCC Rules, which provide reasonable protection against such interference in a residential installation. In the unlikely event that there is interference to radio or television reception (which can be determined by turning the HP 48 off and on or by removing the batteries), try the following:

- Reorienting the receiving antenna.
- Relocating the calculator with respect to the receiver.

For more information, consult your dealer, an experienced radio/television technician, or the following booklet, prepared by the Federal Communications Commission: How to Identify and Resolve Radio-TV Interference Problems. This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock Number 004-000-00345-4. At the first printing of this manual, the telephone number was (202) 783-3238.

West Germany. This is to certify that this equipment is in accordance with the Radio Interference Requirements of Directive FTZ 1046/84. The German Bundespost was notified that this equipment was put into circulation, the right to check the serie for compliance with the requirements was granted.

# Messages

This appendix lists selected HP 48 messages.

In the following tables, messages are first arranged alphabetically by name and then numerically by message number.

#### **Messages Listed Alphabetically**

Message	Meaning	# (hex)
Acknowledged	Alarm acknowledged.	619
Autoscaling	Calculator is autoscaling x-and/or y-axis.	610
Awaiting Server Cmd.	Indicates Server mode active.	COC
Bad Argument Type	One or more stack arguments were incorrect type for operation.	202
Bad Argument Value	Argument value out of operation's range.	203
Bad Guess(es)	Guess(es) supplied to HP Solve application or ROOT lie outside domain of equation.	A01

Message	Meaning	# (hex)
Bad Packet Block check	Computed packet checksum doesn't match checksum in packet.	C01
Can't Edit Null Char.	Attempted to edit a string containing character "0".	102
Circular Reference	Attempted to store a variable name into itself.	129
Connecting	Indicates verifying IR or serial connection.	C0A
Constant?	HP Solve application or ROOT returned same value at every sample point of current equation.	A02
Copied to stack	→STK copied selected equation to stack.	623
Current equation:	Identifies current equation.	608
Deleting Column	MatrixWriter application is deleting a column.	504
Deleting Row	MatrixWriter application is deleting a row.	503
Directory Not Allowed	Name of existing directory variable used as argument.	12A
Directory Recursion	Attempted to store a directory into itself.	002
Empty catalog	No data in current catalog (Equation, Statistics, Alarm)	60D

Message	Meaning	# (hex)
Enter alarm, press SET	Alarm entry prompt.	61A
Enter eqn, press NEW	Store new equation in EQ.	60A
Enter value (zoom out if >1), press ENTER	Zoom operations prompt.	622
Extremum	Result returned by HP Solve application or ROOT is an extremum rather than a root.	A06
HALT Not Allowed	A program containing HALT executed while MatrixWriter application, DRAW, or HP Solve application active.	126
I/O setup menu	Identifies I/O setup menu.	61C
Implicit () off	Implicit parentheses off.	207
Implicit () on	Implicit parentheses on.	208
Incomplete Subexpression	▶, ▼, or ENTER pressed before all function arguments supplied.	206
Inconsistent Units	Attempted unit conversion with incompatible units.	B02
Infinite Result	Math exception: Calculation such as 1/0 infinite result.	305
Inserting Column	MatrixWriter application is inserting a column.	504

Messa	age	Meaning	# (hex)
Inserting	Row	MatrixWriter application is inserting a row.	503
Insufficie Memory	nt	Not enough free memory to execute operation.	001
Insufficie Data	nt Σ	A Statistics command was executed when ΣDAT did not contain enough data points for calculation.	603
Interrupte	d	The HP Solve application or ROOT was interrupted by ATTN.	A03
Invalid Ar Element	ray	ENTER returned object of wrong type for current matrix.	502
Invalid Ca	rd Data	HP 48 does not recognize data on plug-in card.	008
Invalid Da	te	Date argument not real number in correct format, or was out of range.	D01
Invalid De	finition	Incorrect structure of equation argument for DEFINE.	12C
Invalid Di	mension	Array argument had wrong dimensions.	501

Message	Meaning	# (hex)
Invalid EQ	Attempted operation from GRAPHICS FCN menu when EQ did not contain algebraic, or, attempted DRAW with CONIC plot type when EQ did not contain algebraic.	607
Invalid IOPAR	IOPAR not a list, or one or more objects in list missing or invalid.	C12
Invalid Name	Received illegal filename, or server asked to send illegal filename.	C17
Invalid PPAR	PPAR not a list, or one or more objects in list missing or invalid.	12E
Invalid PRTPAR	PRTPAR not a list, or one or more objects in list missing or invalid.	C13
Invalid PTYPE	Plot type invalid for current equation.	620
Invalid Repeat	Alarm repeat interval out of range.	D03
Invalid Server Cmd.	Invalid command received while in Server mode.	C08
Invalid Syntax	HP 48 unable execute ENTER or STR→ due to invalid object syntax.	106

Message	Meaning	# (hex)
Invalid Time	Time argument not real number in correct format, or out of range.	D02
Invalid Unit	Unit operation attempted with invalid or undefined user unit.	B01
Invalid User Function	Type or structure of object executed as user-defined function was incorrect.	103
Invalid Σ Data	Statistics command executed with invalid object stored in \$\infty DAT\$.	601
Invalid Σ Data LN(Neg)	Non-linear curve fit attempted when ΣDAT matrix contained a negative element.	605
Invalid Σ Data LN(0)	Non-linear curve fit attempted when ΣDAT matrix contained a 0 element.	606
Invalid ΣPAR	ΣPAR not list, or one or more objects in list missing or invalid.	604
LAST CMD Disabled	(LAST CMD) pressed while that recovery feature disabled.	125
LAST STACK Disabled	[LAST STACK] pressed while that recovery feature disabled.	124
LASTARG Disabled	LASTARG executed while that recovery feature disabled.	205

Message	Meaning	# (hex)
Low Battery	System batteries too low to safely print or perform I/O.	C14
Memory Clear	HP 48 memory was cleared.	005
Name Conflict	Execution of   (where) attempted to assign value to variable of integration or summation index.	13C
Name the equation, press ENTER	Name equation and store it in EQ.	60B
Name the stat data, press ENTER	Name statistics data and store it in \$\infty DAT.	621
Negative Underflow	Math exception: Calculation returned negative, non-zero result greater than -MINR.	302
No Current Equation	SOLVR, DRAW, or RCEQ executed with nonexistent <i>EQ</i> .	104
No current equation	Plot and HP Solve application status message.	609
No Room in Port	Insufficient free memory in specified RAM port.	00B
No Room to Save Stack	Not enough free memory to save copy of the stack. LAST STACK is automatically disabled.	101

Message	Meaning	# (hex)
No Room to Show Stack	Stack objects displayed by type only due to low memory condition.	131
No stat data to plot	No data stored in $\Sigma DAT$ .	60F
Non-Empty Directory	Attempted to purge non- empty directory.	12B
Non-Real Result	Execution of HP Solve application, ROOT, DRAW, or freturned result other than real number or unit.	12F
Nonexistent Alarm	Alarm list did not contain alarm specified by alarm command.	D04
Nonexistent ΣDAT	Statistics command executed when \$\infty DAT\$ did not exist.	602
Object Discarded	Sender sent an EOF (Z) packet with a "D" in the data field.	COF
Object In Use	Attempted PURGE or STO into a backup object when its stored object was in use.	009
Object Not in Port	Attempted to access a nonexistent backup object or library.	00C
(OFF SCREEN)	Function value, root, extremum, or intersection was not visible in current display.	61F

Message	Meaning	# (hex)
Out of Memory	One or more objects must be purged to continue calculator operation.	135
Overflow	Math exception: Calculation returned result greater in absolute value than MAXR.	303
Packet #	Indicates packet number during send or receive.	C10
Parity Error	Received bytes' parity bit doesn't match current parity setting.	C05
Port Closed	Possible I/R or serial hardware failure. Run self-test.	C09
Port Not Available	Used a port command on an empty port, or one containing ROM instead of RAM.	00A
	Attempted to execute a server command that itself uses the I/O port.	
Positive Underflow	Math exception: Calculation returned positive, non-zero result less than MINR.	301
Power Lost	Calculator turned on following a power loss. Memory may have been corrupted.	006

Message	Meaning	# (hex)
Processing Command	Indicates processing of host command packet.	C11
Protocol Error	Received a packet whose length was shorter than a null packet.	C07
	Maximum packet length parameter from other machine is illegal.	128 1000000
Receive Buffer Overrun	Kermit: More than 255 bytes of retries sent before HP 48 received another packet.	C04
	SRECV: Incoming data overflowed the buffer.	
Receive Error	UART overrun or framing error.	C03
Receiving	Identifies object name while receiving.	C0E
Retry #	Indicates number of retries while retrying packet exchange.	C0B
Select a model	Select statistics curve fitting model.	614
Select plot type	Select plot type.	60C
Select repeat interval	Select alarm repeat interval.	61B

B: Messages

Message	Meaning	# (hex)
Sending	Identifies object name while sending.	COD
Sign Reversal	HP Solve application or ROOT unable to find point at which current equation evaluates to zero, but did find two neighboring points at which equation changed sign.	A05
Timeout	Printing to serial port: Received XOFF and timed out waiting for XON.	C02
	Kermit: Timed out waiting for packet to arrive.	
Too Few Arguments	Command required more arguments than were available on stack.	201
Transfer Failed	10 successive attempts to receive a good packet were unsuccessful.	C06
Unable to Isolate	ISOL failed because specified name absent or contained in argument of function with no inverse.	130
Undefined Local Name	Executed or recalled local name for which corresponding local variable did not exist.	003

Message	Meaning	# (hex)
Undefined Name	Executed or recalled global name for which corresponding variable does not exist.	204
Undefined Result	Calculation such as 0/0 generated mathematically undefined result.	304
Undefined XLIB Name	Executed an XLIB name when specified library absent.	004
Wrong Argument Count	User-defined function evaluated with an incorrect number of parenthetical arguments.	128
x and y-axis zoom.	Identifies zoom option.	627
x axis zoom.	Identifies zoom option.	625
x axis zoom w∕AUTO.	Identifies zoom option.	624
y axis zoom.	Identifies zoom option.	626
ZERO	Result returned by the HP Solve application or ROOT is a root (a point at which current equation evaluates to zero).	A04
11 11	Identifies no execution action when EXECS pressed.	61E

688 B: Messages

## Messages Listed Numerically

# (hex)	Message	
	General Messages	
001	Insufficient Memory	DE L
002	Directory Recursion	
003	Undefined Local Name	
004	Undefined XLIB Name	
005	Memory Clear	
006	Power Lost	
800	Invalid Card Data	
009	Object In use	
00A	Port Not available	
00B	No Room in Port	
00C	Object Not in Port	
101	No Room to Save Stack	
102	Can't Edit Null Char.	
103	Invalid User Function	
104	No Current Equation	
106	Invalid Syntax	
124	LAST STACK Disabled	
125	LAST CMD Disabled	
126	HALT Not Allowed	
128	Wrong Argument Count	
129	Circular Reference	
12A	Directory Not Allowed	
12B	Non-Empty Directory	
12C	Invalid Definition	
12E	Invalid PPAR	
12F	Non-Real Result	

# (hex)	Message
unuel and	General Messages (continued)
130	Unable to Isolate
131	No Room to Show Stack
	Out-of-Memory Prompts
135	Out of Memory
13C	Name Conflict
<del></del>	Stack Errors
201	Too Few Arguments
202	Bad Argument Type
203	Bad Argument Value
204	Undefined Name
205	LASTARG Disabled
	EquationWriter Application Messages
206	Incomplete Subexpression
207	Implicit () off
208	Implicit () on
- 3U73.	Floating-Point Errors
301	Positive Underflow
302	Negative Underflow
303	Overflow
304	Undefined Result
305	Infinite Result
	Array Messages
501	Invalid Dimension
502	Invalid Array Element
503	Deleting Row
504	Deleting Column
505	Inserting Row

# (hex)	Message	50 to
	Array Messages (continued)	
506	Inserting Column	41.7
	Statistics Messages	
601	Invalid Σ Data	
602	Nonexistent ZDAT	
603	Insufficient Σ Data	
604	Invalid XPAR	
605	Invalid Σ Data LN(Neg)	
606	Invalid Σ Data LN(0)	
Plot, i	O, Time and HP Solve Application Messag	ges
607	Invalid EQ	1000
608	Current equation:	
609	No current equation.	
60A	Enter eqn, press NEW	
60B	Name the equation, press ENTER	
60C	Select plot type	
60D	Empty catalog	
60F	No Statistics data to plot	
610	Autoscaling	
614	Select a model	
619	Acknowledged	
61A	Enter alarm, press SET	
61B	Select repeat interval	
61C	I/O setup menu	
61D	Plot type:	
61E	in a control of the control	
61F	(OFF SCREEN)	
620	Invalid PTYPE	
621	Name the stat data, press ENTER	

# (hex)	Message	
	Application Messages (continued)	
622	Enter value (zoom out if >1), press ENTER	
623	Copied to stack	
624	x axis zoom w/AUTO.	
625	x axis zoom.	
626	y axis zoom.	
627	x and y-axis zoom.	
A01	Bad Guess(es)	
A02	Constant?	
A03	Interrupted	
A04	Zero	
A05	Sign Reversal	
A06	Extremum	
	Unit Management	_
B01	Invalid Unit	Π
B02	Inconsistent Units	

# (hex)	Message
	I/O and Printing
C01	Bad Packet Block check
C02	Timeout
C03	Receive Error
C04	Receive Buffer Overrun
C05	Parity Error
C06	Transfer Failed
C07	Protocol Error
C08	Invalid Server Cmd
C09	Port Closed
C0A	Connecting
C0B	Retry #
COC	Awaiting Server Cmd.
COD	Sending
C0E	Receiving
C0F	Object Discarded
C10	Packet #
C11	Processing Command
C12	Invalid IOPAR
C13	Invalid PRTPAR
C14	I/O: Batt Too Low
C15	Empty Stack
C17	Invalid Name
	Time Messages
D01	Invalid Date
D02	Invalid Time
D03	Invalid Repeat
D04	Nonexistent Alarm

## **HP 48 Character Codes**

Most of the characters in the HP 48 character set can be directly typed into the display from the Alpha keyboard. For example, to display \$, type  $\alpha$   $\blacktriangleleft$  4. (The Alpha keyboard is presented in chapter 2.) Any character in the set can be displayed by typing its corresponding character code and then executing the CHR command. The syntax is char# CHR. Certain characters in the set are not on the Alpha keyboard. To display one of these characters, you must type its character code and execute CHR.

The character tables on the following pages show the HP 48 characters and their corresponding character codes. (This set, except for character numbers 128 through 159, is based on the ISO 8859 Latin 1 character set.)



If you find that a character you frequently use is not available on the primary or alpha keyboards (see chapter 2) for all the available characters), you can assign that character to the user keyboard for easy access. See "Making User Key Assignments" on page 217 for more information.

## Character Codes (0 — 127)

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
0		32		64	6	96	
1		33	!	65	А	97	a
2		34	11	66	В	98	ь
3		35	#	67	С	99	C
4		36	\$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	1	71	G	103	9
8	=	40		72	Н	104	h
9		41		73	I	105	i
10		42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	1
13		45	-	77	M	109	m
14		46		78	N	110	п
15		47	1	79	0	111	0
16		48	0	80	P	112	Р
17		49	1	81	Q	113	q
18		50	2	82	R	114	r
19		51	3	83	S	115	S
20		52	4	84	T	116	t
21		53	5	85	U	117	u
22		54	6	86	٧	118	V
23		55	7	87	М	119	(J)
24		56	8	88	X	120	×
25		57	9	89	Y	121	y
26		58	:	90	Z	122	Z
27		59	;	91	[	123	<
28		60	<	92	1	124	
29		61	=	93	]	125	>
30		62	>	94	^	126	~
31	- 4	63	?	95	_	127	

#### Character Codes (128 — 255)

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
128	4	160		192	À	224	à
129	X	161	i	193	Á	225	á
130	⊽	162	<b>#</b>	194	Å	226	å
131	1	163	£	195	Ä	227	ä
132	Ĵ	164	Ŏ	196	Ä	228	ä
133	Σ	165	¥	197	A	229	a
134		166	1	198	Æ	230	22
135	π	167	9	199	Ç	231	9
136	9	168	**	200	È	232	è
137	≟	169	9	201	É	233	é
138	$\geq$	170	à	202	Ê	234	é
139	<b>≠</b>	171	«	203	Ë	235	ë
140	oc	172	~	204	Ì	236	ì
141	÷	173	***	205	Í	237	í
142	€	174	8	206	Î	238	î
143	ψ	175	- I	207	ï	239	ï
144	<b>小</b>	176	D	208	Ð	240	đ
145	Υ	177	±	209	Ñ	241	ñ
146	δ	178	5	210	ò	242	ò
147	€	179	3	211	ó	243	ó
148	$\gamma_{l}$	180	*	212	Ô	244	ô
149	8	181	μ	213	ő	245	ő
150	X	182	17	214	ö	246	ö
151	ρ	183		215	×	247	÷
152	σ	184	3	216	ø	248	ø
153	τ	185	1	217	Ù	249	ù
154	ω	186	9	218	Ú	250	ú
155	Δ	187	>	219	û	251	û
156	Π	188	4	220	Ü	252	ü
157	Ω	189	14	221	Ý	253	ý
158		190	34	222	Þ	254	Þ
159	00	191	٤	223	β	255	ÿ

# **Menu Numbers**

The following table lists the HP 48 built-in menus and the corresponding menu numbers.

Menu #	Menu Name	Menu #	Menu Name
0	Last Menu	19	I/O SETUP
1	CST	20	MODES
2	VAR	21	MODES Customization
3	MTH	22	MEMORY
4	MTH PARTS	23	MEMORY Arithmetic
5	MTH PROB	24	LIBRARY
6	MTH HYP	25	PORT 0
7	MTH MATR	26	PORT 1
8	MTH VECTR	27	PORT 2
9	MTH BASE	28	EDIT
10	PRG	29	SOLVE
11	PRG STK	30	SOLVE SOLVR
12	PRG OBJ	31	PLOT
13	PRG DISP	32	PLOT PTYPE
14	PRG CTRL	33	PLOT PLOTR
15	PRG BRCH	34	ALGEBRA
16	PRG TEST	35	TIME
17	PRINT	36	TIME ADJST
18	1/0	37	TIME ALRM

Menu #	Menu Name	Menu #	Menu Name
38	TIME ALRM RPT	49	UNITS FORCE
39	TIME SET	50	UNITS ENRG
40	STAT	51	UNITS POWR
41	STAT MODL	52	UNITS PRESS
42	UNITS Catalog	53	UNITS TEMP
43	UNITS LENG	54	UNITS ELEC
44	UNITS AREA	55	UNITS ANGL
45	UNITS VOL	56	UNITS LIGHT
46	UNITS TIME	57	UNITS RAD
47	UNITS SPEED	58	UNITS VISC
48	UNITS MASS	59	UNITS Command

# **Listing of HP 48 System Flags**

This appendix lists the HP 48 system flags in functional groups. All flags can be set, cleared, and tested. The default state of the flags is *clear*, except for the Binary Integer Wordsize flags (flags -5 through -10).

#### System Flags

Flag	Description
	Symbolic Math Flags
-1	Principal Solution.  Clear: QUAD and ISOL return a result representing all possible solutions.  Set: QUAD and ISOL return only the principal solution.
-2	Symbolic Constants. Clear: Symbolic constants (e, i, $\pi$ , MAXR, and MINR) retain their symbolic form when evaluated, unless the Numerical Results flag $-3$ is set. Set: Symbolic constants evaluate to numbers, regardless of the state of the Numerical Results flag $-3$ .
-3	Numerical Results.  Clear: Functions with symbolic arguments, including symbolic constants, evaluate to symbolic results.  Set: Functions with symbolic arguments, including symbolic constants, evaluate to numbers.
-4	Not used.

Flag	Description				
	Binary Integer Math Flags				
-5	Binary Integer Wordsize.				
thru	Combined states of flags -5 through -10 set the wordsize from 1 to 64 bits.				
-10					
	Binary Integer Base.				
-11	HEX: -11 set, -12 set.				
and	DEC: -11 clear, -12 clear.				
-12	OCT: -11 set, -12 clear.				
	BIN: -11 clear, -12 set.				
-13	1903				
and	Not used.				
-14					
	Coordinate System Flags				
-15	Rectangular: -15 clear, -16 clear.				
and	Polar/Cylindrical: -15 clear, -16 set.				
-16	Polar/Spherical: -15 set, -16 set.				
	Trigonometric Angle Mode Flags				
-17	Degrees: -17 clear, -18 clear.				
and	Radians: -17 set, -18 clear.				
-18	Grads: -17 clear, -18 set.				
	Complex Mode Flag				
-19	Clear:→V2 and → 2D create a 2-dimensional vector from 2 real numbers.				
20 20	Set:→V2 and → 2D create a complex number from 2 real numbers.				

Flag	Description
	Math Exception-Handling Flags
-20	Underflow Exception.
	Clear: Underflow exception returns 0.
less.	Set: Underflow exception treated as an error.
-21	Overflow Exception.
	Clear: Overflow exception returns ±9.9999999999E499.
	Set: Overflow exception treated as an error.
-22	Infinite Result Exception.
	Clear: Infinite result exception treated as an error.
	Set: Infinite result exception returns ±9.99999999999E499.
-23	Negative Underflow Indicator.
-24	Positive Underflow Indicator.
-25	Overflow Indicator.
-26	Infinite Result Indicator.
	When an exception occurs, corresponding flag (-23
	through -26) is set, regardless of whether or not the
	exception is treated as an error.
-27	
thru	Not used.
-29	

Flag	Description		
Plotting and Graphics Flags			
-30	Function Plotting.  Clear: For equations of form $y = f(x)$ , only $f(x)$ is drawn.  Set: For equations of form $y = f(x)$ , separate plots of $y$ and $f(x)$ are drawn.		
-31	Curve Filling.  Clear: Curve filling between plotted points enabled.  Set: Curve filling between plotted points suppressed.		
-32	Graphics Cursor.  Clear: Graphics cursor always dark.  Set: Graphics cursor dark on light background and light on dark background.		
	I/O and Printing Flags		
-33	I/O Device.  Clear: I/O directed to serial port.  Set: I/O directed to IR port.		
-34	Printing Device.  Clear: Printer output directed to IR printer.  Set: Printer output directed to serial port if flag -33 is clear.		
-35	I/O Data Format.  Clear: Objects transmitted in ASCII form.  Set: Objects transmitted in memory image form.		
-36	RECV Overwrite.  Clear: If file name received by HP 48 matches existing HP 48 variable name, new variable name with number extension is created to prevent overwrite.  Set: If file name received by HP 48 matches existing HP 48 variable name, existing variable is overwritten.		

Flag	Description
	I/O and Printing Flags (continued)
-37	Double-Spaced Printing.
	Clear: Single-spaced printing.
	Set: Double-spaced printing.
-38	Linefeed.
	Clear: Linefeed added at end of each print line.
	Set: No linefeed added at end of each print line.
-39	I/O Messages.
	Clear: I/O messages displayed.
	Set: I/O messages suppressed.
	Time Management Flags
-40	Clock Display.
	Clear: Ticking clock displayed only when TIME menu
82	selected.
11.2	Set: Ticking clock displayed at all times.
-41	Clock Format.
	Clear: 12-hour clock.
	Set: 24-hour clock.
-42	Date Format.
	Clear: MM/DD/YY (month/day/year) format.
	Set: DD.MM.YY (day.month.year) format.
-43	Repeat Alarms Not Rescheduled.
	Clear: Unacknowledged repeat appointment alarms automatically rescheduled.
	Set: Unacknowledged repeat appointment alarms not rescheduled.

Flag	Description			
	Time Management Flags (continued)			
-44	Acknowledged Alarms Saved.			
	Clear: Acknowledged appointment alarms deleted from alarm list.			
	Set: Acknowledged appointment alarms saved in alarm list.			
	Display Format Flags			
-45	Number of Decimal Digits.			
thru	Combined states of flags -45 through -48 sets number of decimal digits in Fix, Scientific, and Engineering modes.			
-48				
	Number Display Format.			
-49	Standard: -49 clear, -50 clear.			
and	Fix: -49 set, -50 clear.			
-50	Scientific: -49 clear, -50 set.			
	Engineering: -49 set, -50 set.			
-51	Fraction Mark.			
	Clear: Fraction mark is . (period).			
	Set: Fraction mark is , (comma).			
-52	Single-Line Display.			
	Clear: Display gives preference to object in level 1, using up to four lines of stack display.			
	Set: Display of object in level 1 restricted to one line.			
-53	Precedence.			
	Clear: Certain parentheses in algebraic expressions suppressed to improve legibility.			
	Set: All parentheses in algebraic expressions displayed.			
-54	Not used.			

Flag	Description		
	Miscellaneous Flags		
-55	Last Arguments.		
	Clear: Operation arguments saved.		
	Set: Operation arguments not saved.		
-56	Error Beep.		
	Clear: Error and BEEP-command beeps enabled.		
	Set: Error and BEEP-command beeps suppressed.		
-57	Alarm Beep.		
	Clear: Alarm beep enabled.		
	Set: Alarm beep suppressed.		
-58	Verbose Messages.		
	Clear: Prompt messages and data automatically displayed.		
	Set: Automatic display of prompt messages and data		
	suppressed.		
-59	Fast Catalog Display.		
	Clear: Equation Catalog (and messages in SOLVE, SOLVR, PLOT, and PLOTR menus) show equation and equation		
	name.		
	Set: Equation Catalog (and messages in SOLVE, SOLVR, PLOT, and PLOTR menus) show equation name only.		
-60	Alpha Lock.		
	Clear: Alpha lock activated by pressing a twice.		
	Set: Alpha lock activated by pressing a once.		
-61	User-Mode Lock.		
	Clear: 1-User mode activated by pressing  usn once.		
	User mode activated by pressing  wice.		
	Set: User mode activated by pressing  use once.		

Flag	Description
	Miscellaneous Flags (continued)
-62	User Mode.  Clear: User mode not active.  Set: User mode active.
-63	Vectored ENTER.  Clear: ENTER evaluates command line.  Set: User-defined ENTER activated.
-64	Index Wrap Indicator.  Clear: Last execution of GETI or PUTI did not increment index to first element.  Set: Last execution of GETI or PUTI did increment index to first element.

# **Operation Index**

This index contains reference information for all operations in the HP 48. For each operation, this index shows:

Name, Key, or Label. The name, key, or menu label associated with the operation. Operation names appear as keys or menu labels.

**Description.** What the operation does (or its value if a unit).

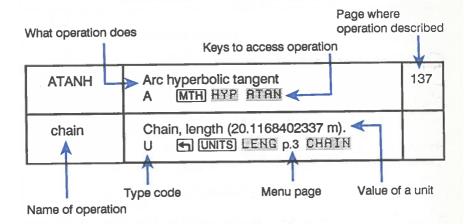
**Type.** The type of operation is given by one of the following codes.

Type Code	Description	
0	Operation. An operation that cannot be included in the command line, in a program, or in an algebraic.	
С	Command. An operation that can be included in programs but <i>not</i> in algebraics.	
F	Function. A command that can be included in algebraics.	
Α	Analytic Function. A function for which the HP 48 provides an inverse and derivative.	
U	Unit.	

**Keys.** The keys to access the operation. Keystroke sequences preceded by "..." can be accessed through more than one menu—to see the keystrokes represented by the "...", refer to the listing in this index for the operation that immediately follows the "...". Operations in multipage menus show the applicable menu page number. Operations that are not key-accessible are identified by "Must be typed in."

Page. Where the operation is described in this manual.

The entries in this index are arranged as follows:



Operations whose names contain both alpha and special characters are listed alphabetically; operations whose names contain special characters only are listed at the end of this index.

Name, Key or Label	Description Type, Keys	Page
а	Are, area (100 m²).  U	
Α	Ampere, electric current (1 A).  U	
Å	Angstrom, length (1 × 10 <sup>-10</sup> m).  U	

Name, Key or Label	Description Type, Keys	Page
444	Associate left.	405
<b>₽</b> + ₽	O ← EQUATION ■ RULES + R  Executes + R until no change in subexpression.  O ← EQUATION ■ RULES  → + R	410
8.9	Associate right.	405
r A→	O SEQUATION RULES ARE Executes R+ until no change in subexpression. O SEQUATION RULES R+	410
ABS	Absolute value.  MTH PARTS ABS  MTH MATR p.2 ABS  F MTH VECTR ABS	148
ACK	Acknowledges displayed past due alarm.  C	447
ACKALL	Acknowledges all past due alarms.  C	447

Name, Key or Label	Description Type, Keys	Page
ACOS	Arc cosine. A (A) (ACOS)	140
ACOSH	Arc hyperbolic cosine. A MTH HYP RCOSH	137
acre	Acre, area (4046.87260987 m²).  U ———————————————————————————————————	
ADJUST	Selects TIME ADJST (adjust) menu.  O  TIME ADJST	634
AF	Add fractions.  O	409
(ALGEBRA)	Selects ALGEBRA menu. O [4][ALGEBRA]	
[ ALGEBRA]	Selects Equation Catalog.  O ♠ ALGEBRA	259
ALOG	Common (base 10) antilogarithm.  A 10 <sup>2</sup>	137

Name, Key or Label	Description Type, Keys	Page
MAJES	Selects TIME ALRM (alarm) menu.  O	
AND	Logical or binary AND.  MTH BASE p.4 AND  F PRG TEST AND	210 493
ANGL	Selects UNITS ANGL menu. O — [UNITS] p.3 RNGL	
APPLY	Returns evaluated expression(s) as argument(s) to unevaluated local name.  F	-
ARC	Draws arc in <i>PICT</i> from $\theta_1$ to $\theta_2$ with center at $(x,y)$ and radius $r$ .  C PRG DSPL ARC	339
ARCHIVE	Makes backup copy of HOME directory.  C ← MEMORY p.3 RRCHI	648
arcmin	Minute of arc, plane angle. (4.62962962963 × 10 <sup>-5</sup> ) U (INITS) p.3 ANGL RECMI	
arcs	Second of arc, plane angle. (7.71604938272 × 10 <sup>-7</sup> )  U — UNITS p.3 ANGL ARCS	

Name, Key or Label	Description Type, Keys	Page
AREA	Calculates and displays area under function graph between two x-values specified by the mark and cursor; returns area to stack.  O FCN AREA	308
arasa.	Selects UNITS AREA menu. O  (1) UNITS	
ARG	Returns polar angle θ. F [MTH] PARTS ARG	166
ARG	Enables/disables LASTARG recovery.  O	221
ARRY→	Returns array elements to stack.  C Must be typed in.	
→ARRY	Combines numbers into array.  C PRG 08J >ARR	90
ASM	Switches between ASCII and binary mode.  O	617
ASIN	Arc sine. A ← ASIN	140
ASINH	Arc hyperbolic sine.  A MTH HYP ASINH	137

Name, Key or Label	Description Type, Keys	Page
ASN	Makes a single user-key assignment.  C MODES ASN	217
ASR	1-bit arithmetic shift right. C MTH BASE p.3 ASR	210
ATAN	Arc tangent. A ← ATAN	140
ATANH	Arc hyperbolic tangent.  A MTH HYP STAN	137
atm	Atmosphere, pressure (101325 kg/m·s²) U ← UNITS p.2 PRESS ATM	
ATTACH	Attaches specified library to current directory.  C  MEMORY p.2 ATTAC	651
ATTN (ON)	Aborts program execution, aborts command line; exits special environments; clears messages.  O ON	54
AU	Astronomical unit, length (1.495979 × 10 <sup>11</sup> m).  U  UNITS LENG p.2 RU	

Name, Key or Label	Description Type, Keys	Page
AUTO	Scales y-axis PLOTR RUTO C → PLOT RUTO	293
AUTO	Scales y-axis; then plots equation PLOTR AUTO O PLOT AUTO	295
AXES	Sets specified coordinates of axes intersection; stores labels PLOTR p.3 RXES C PPLOT p.3 RXES	320
AXES	Recalls axes intersection to stack PLOTR p.3 PAXES O PLOT p.3 AXES	319
A/PM	Switches clock between AM and PM.  O SET AZPM Switches alarm time between AM and PM.  O SITIME ALRM AZPM	442
b	Barn, area (1 × 10 <sup>-28</sup> m²).  U	
bar	Bar, pressure (100000 kg/m·s²).  U	
BAR	Selects BAR plot type. C PTYPE BAR	328

Name, Key or Label	Description Type, Keys	Page
BARPLOT	Draws bar plot of data in ∑DAT.  C ← STAT p.3 BARPL	378
BRSE	Selects MTH BASE menu. O MTH BASE	
BAUD	Sets one of four available baud rates.  C	617
bbl	Barrel, volume (.158987294928 m³). U ← UNITS VOL p.4 BBL	
BEEP	Sounds beep. C PRG CTRL p.3 BEEP	523
<b>2)</b>	Enables/disables error BEEP.  O	221
BESTFIT	Selects statistics model yielding largest correlation coefficient (absolute value) and executes LR.  C	377
BIN	Sets binary base.  MTH BASE BIN  C  MODES p.4 BIN	208
BINS	Sorts elements in independent variable column of ΣDAT into N + 2 bins (up to a maximum of 1048573 bins).  C STAT p.2 BINS	382

Name, Key or Label	Description Type, Keys	Page
BLANK	Creates blank graphics object. C PRG DSPL p.3 BLAN	343
вох	Draws box with opposite corners defined by specified coordinates.  C [PRG] DSPL BOX	339
BOX	Draws box with opposite corners defined by mark and cursor.  DRAW p.2 BOX  AUTO p.2 BOX  O 4-GRAPH p.2 BOX	337
Bq	Becquerel, activity (1 1/s).  U — UNITS p.3 RAD BQ	
BRCH	Selects PRG BRCH (program branch) menu.  O [PRG] BRCH	
Btu	International Table Btu, energy (1055.05585262 kg·m²/s²)  U — UNITS p.2 ENRG BTU	
bu	Bushel, volume (.03523907 m³).  U ← UNITS VOL p.4 BU	
BUFLEN	Returns number of characters in serial buffer.  C	634

Name, Key or Label	Description Type, Keys	Page
BYTES	Returns object size (in bytes) and checksum for object.  C MEMORY BYTES	101
B→R	Binary-to-real conversion.  C MTH BRSE p.2 B→R	210
С	Speed of light (299792458 m/s). U 4 UNITS SPEED p.2 C	
С	Coulomb, electric charge (1 A·s).  U	
°C	Degrees Celsius, temperature.  U	
cal	Calorie, energy (4.186 kg·m²/s²) U	
CASE	Begins CASE structure. C [PRG] BRCH CASE	498
← CASE	Types CASE THEN END END.  O PRG BRCH  CASE	498
→ CASE	Types THEN END.  O PRG BRCH CASE	498

Name, Key or Label	Description Type, Keys	Page
CAT	Selects Equation Catalog.  [4] PLOT CAT  [4] SOLVE CAT	259
	O ALGEBRA Selects STAT Catalog. O STATI CRT	370
	O STAT CAT Selects Alarm Catalog.  TIME CAT O PIME	449
cd	Candela, luminous intensity (1 cd).  U	
CEIL	Returns next greater integer.  F MTH PARTS p.3 CEIL	148
CENT	Redraws graph with center at cursor position.  DRAW CENT	302
	O GRAPH CENT	

Name, Key or Label	Description Type, Keys	Page
CENTR	Sets center of plot display at specified (x, y) coordinates PLOTR p.2 CENT	295
← CENT	C PLOT p.2 CENT  Recalls plot-center coordinates to stack.  PLOTR p.2 PCENT  O PLOT p.2 CENT	293
CF 48	Clears specified flag.  PRG TEST p.3 CF  C → MODES p.2 CF	516
%CH	Returns % change from level 2 to level 1.  F MTH PRRTS p.2 2CH	138
chain	Chain, length (20.1168402337 m). U   UNITS LENG p.3 CHAIN	
CHR	Converts character code to one-character string.  C [PRG] [UBJ] p.3 [CHR]	90
Ci	Curie, activity (3.7 × 10 <sup>10</sup> 1/s).  U	

Name, Key or Label	Description Type, Keys	Page
OTREL	Draws circle with center at the mark and radius equal to the distance from cursor to mark.  DRAW p.2 CIRCL BUTO p.2 CIRCL O GRAPH p.2 CIRCL	337
CKSM	Selects one of three available checksum error-detect schemes.  C	618
CLEAR	Clears stack. C → CLR	64
CLR	In EquationWriter entry mode, clears screen.  O ⑤ EQUATION ⓒ CLR  Clears PICT.  DRAW ⓒ CLR  AUTO ⓒ CLR  O ⑤ GRAPH ⓒ CLR	303
CLK	Switches ticking clock display on and off.  O ← MODES p.2 CLK	221

Name, Key or Label	Description Type, Keys	Page
CLKADJ	Adds specified number of clock ticks to system time.  C TIME ADJST p.2 CLKR	443
CLLCD	Blanks stack display. C PRG DSPL p.4 CLLCD	520
CLOSEIO	Closes I/O port. C	615
CLE	Purges statistical data in ΣDAT.  C (STAT) CLΣ	368
CLUSR	Purges all user variables. C Must be typed in.	
CLVAR	Purges all user variables. C → PURGE	115
cm	Centimeter, length (.01 m). U	
CMD	Enables/disables last command line recovery.  O ( MODES) p.2 CMD	221
cm^2	Square centimeter, area (1 × 10 <sup>-4</sup> m²).  U — UNITS AREA CM^2	

Name, Key or Label	Description Type, Keys	Page
cm^3	Cubic centimeter, volume (1 × 10 <sup>-6</sup> m³).  U	
cm/s	Centimeters per second, speed (.01 m/s).  U — UNITS SPEED CMZS	
<b>P</b> 3( <b>0</b> )1	Switches curve filling on and off.  O ← MODES p.2 CNCT	221
CNRM	Calculates column norm of array.  C MTH MATR p.2 CNRM	359
+CGL	Inserts a row of zeros at current column in MatrixWriter application.  O ► MATRIX p.2 +COL	351
<b>-6</b> 01	Deletes current column in MatrixWriter application.  O ► MATRIX p.2 - COL	351
COLCT	Collects like terms in expression.  C ← ALGEBRA COLCT	395
COLCT	Collects like terms in specified subexpression.  O ← EQUATION ■ RULES COLCT	402
COLE	Specifies dependent and independent columns in ΣDAT.  C Must be typed in.	

Name, Key or Label	Description Type, Keys	Page
СОМВ	Returns number of combinations of <i>n</i> items taken <i>m</i> at a time.  F MTH PROB COMB	147
CON	Creates constant array. C MTH MATR CON	359
CONIC	Selects CONIC plot type. C PTYPE CONIC	.327
CONJ	Returns complex conjugate.  F MTH PARTS CONJ	.166
CONT	Continues halted program.  C	520
CONVERT	Converts unit object to dimensions of specified compatible unit.  C PUNITS CONV	194
COORD	Displays cursor coordinates at bottom left of display.  DRAW COORD  RUTO COORD  O GRAPH COORD	302

Name, Key or Label	Description Type, Keys	Page
CORR	Calculates correlation coefficient of statistical data in ΣDAT.  C STAT p.4 CORR	377
COS	Cosine. A [COS]	140
COSH	Hyperbolic cosine.  A MTH HYP COSH	137
COV	Calculates covariance of statistical data in ∑DAT. C ← STAT p.4 COV	377
CR	Causes printer to do carriage return/line feed.  C	608
CRDIR	Creates a directory.  C	120
CROSS	Cross product of 2- or 3-element vector.  C MTH VECTR CROSS	353
CST	Selects CST (custom) menu. O CST	212
ESI	Returns contents of CST variable.  O MODES CST	213

Name, Key or Label	Description Type, Keys	Page
ct	Carat, mass (.0002 kg). U	
CTRL	Selects PRG CTRL (program control) menu.  O PRG CTRL	
cu	US cup, volume (2.365882365 × 10 <sup>-4</sup> m³).  U	
C→PX	Converts user-unit coordinates to pixel coordinates.  C PRG DSPL p.2 C→PX	324
C→R	Separates complex number into two real numbers.  C [PRG] □BJ pg.2 C→R	91
d	Day, time (86400 s). U	
2D	Assembles or takes apart a complex number or 2D vector.  O 4 2D	160
3D	Assembles or takes apart a 3D vector.  O (**)3D	173

Name, Key or Label	Description Type, Keys	Page
4D	Distribute left.  O ( RULES +D	406
<b>→</b> +D	Executes until no change in subexpression.  O EQUATION _ RULES +D	410
D:4	Distribute right.  O ◀☐EQUATION ◀ RULES D→	406
	ExecutesD+ _ until no change in subexpression.	410
	O SEQUATION RULES  D+	
DATE	Returns system date. C TIME p.2 DATE	455
DATE+	Returns new date from specified date and number of days.  C TIME p.2 DRTE+	454
→DATE	Sets specified system date.  C ← TIME SET → DAT	441
>#\1:14=	Sets specified alarm date.  O  TIME	445
OHY.	Sets alarm repeat interval to n days.  O	445

Name, Key or Label	Description Type, Keys	Page
DBUG	Halts program execution before first object.  O PRG CTRL DBUG	484
DDAYS	Returns number of days between two dates.  C  TIME p.2 DDRYS	455
DEC	Sets decimal base.  MTH BASE DEC  C  MODES p.4 DEC	208
DECR	Decrements value of specified variable.  C [ MEMORY] DECR	513
DEFINE	Creates variable or user-defined function.  C	107 151
→DEF	Expands trigonometric and hyperbolic functions in terms of EXP and LN.  O ( EQUATION RULES > DEF	409
DEG	Sets Degrees mode. C ← MODES p.3 DEG	139
DEL	Deletes character under cursor.  O DEL	75

Name, Key or Label	Description Type, Keys	Page
	Erases area whose opposite corners are defined by mark and cursor.	337
	DRAW p.3 DEL	
	RUTO p.3 DEL	
	O GRAPH p.3 DEL	
(40)5	Deletes all characters from cursor to start of word.	68
	← EDIT +DEL	
	O EDIT +DEL	
€ FDEL	Deletes all characters from cursor to start of line.	68
	STEDIT (+DEL	
	O EDIT A +DEL	
9/2/2-9	Deletes all characters from cursor to start of next word.	68
	← EDIT DEL>	
	O EDIT DEL+	
	Deletes all characters from cursor to end of line.	68
	← EDIT → DEL+	
	O EDIT ADEL+	

Name, Key or Label	Description Type, Keys	Page
DELALARM	Deletes alarm from system alarm list.  C	453
DELAY	Sets delay time between lines sent to printer.  C  PRINT p.2 DELAY	607
DELKEYS	Clears specified user-key assignment.  C MODES DELK	219
DEPND	Specifies name of dependent plot variable PLOTR p.2 DEPN C PLOT p.2 DEPN	318
<b>₽</b> DEPN	Recalls dependent plot variable to stack PLOTR p.2 DEPN O PLOT p.2 DEPN	318
DEPTH	Returns number of objects on stack.  C PRG STK DEPTH	78
DET	Determinant of a matrix.  C MTH MATR DET	360
DETACH	Detaches specified library from current directory.  C  MEMORY p.2 DETAC	653

Name, Key or Label	Description Type, Keys	Page
<b>∄D</b> OJANIA	Double invert.  O ( EQUATION   RULES DINV	401
DISP	Displays object in specified display line.  PRG DSPL p.4 DISP  C PRG CTRL p.2 DISP	523
DNEG	Double negate. O ← EQUATION ← RULES DNEG	400
DO	Begins indefinite loop.  C PRG BRCH DD	510
<b>5</b> 00	Types DO UNTIL END. O PRG BRCH  DO	510
DOERR	Aborts program execution and displays specified message.	546
	C PRG CTRL p.3 DOERR	1774
DOT	Dot product of two vectors.  C MTH VECTR DOT	353

Name, Key or Label	Description Type, Keys	Page
DOT+	Turns on pixels as cursor moves DRAW p.2 DOT+ AUTO p.2 DOT+ O	337
DOT-	Turns off pixels as cursor moves DRAW p.2 DOT AUTO p.2 DOT- O ⑤GRAPH p.2 DOT-	337
DRAW	Plots equation without axes PLOTR DRAW C PPLOT DRAW	292
DRANA	Plots equation with axes PLOTR DRAW O PLOT DRAW	296
DRAX	Draws axes PLOTR p.3 DRAX C PLOT p.3 DRAX	319

Name, Key or Label	Description Type, Keys	Page
DROP	Drops object in level 1; moves all remaining objects down one level.  C	64
DROPN	Drops n objects from stack.  C PRG STK pg.2 DRPN	78
DRPN	Drops all objects from stack at and below pointer.  O *STK p.2 DRPN	71
DROP2	Drops first two objects from stack.  C PRG STK pg.2 DR0P2	78
DSPL	Selects PRG DSPL (program display) menu. O [PRG] DSPL	
DTAG	Removes all tags from object. C PRG OBJ pg.2 DTRG	91
DUP	Duplicates object in level 1. C PRG STK pg.2 DUP	65
DUPN	Duplicates n objects on stack.  C PRG STK pg.2 DUPN	78
DUPN	Duplicates all objects on stack from pointer through stack level 1.  O +STK p.2 DUPN	71

Name, Key or Label	Description Type, Keys	Page
DUP2	Duplicates objects in level 1 and level 2.  C PRG STK pg.2 DUP2	78
dyn	Dyne, force (.00001 kg·m/s²). U — UNITS p.2 FORCE DYN	
D→R	Degrees-to-radians conversion.  F MTH VECTR p.2 D+R	142
е	Symbolic constant e (2.71828182846).	144
ECHO	Copies object in current level to command line.  O +STK ECHO	71
EDEQ	Returns contents of EQ to command line for editing.  • PLOT EDEQ  O • SOLVE EDEQ	256

Name, Key or Label	Description Type, Keys	Page
EDIT	When command line not active, copies level-1 object into command line and selects EDIT menu.	66
	When command line active, selects EDIT menu.	68
	O (SEDIT)	2
- 1- 1- 	Selects EDIT menu.  O (**)[MATRIX] (**)[EDIT]	350
	Returns equation to command line and selects EDIT menu.	242
	O SEQUATION SEDIT	72
5	O FSTK FEDIT	7-2
EDIT	Copies selected equation into command line and selects EDIT menu.  PLOT CRT EDIT  SOLVE CRT EDIT  O PALGEBRA EDIT	259
	Copies subexpression into command line and selects EDIT menu.  O [ EQUATION]	244
	Copies selected matrix to MatrixWriter application.  O 「「STAT」 CAT EDIT	371
	Edits current matrix cell.	351
	O MATRIX EDIT  Displays selected alarm and selects ALRM (alarm) menu.	450
	O STIME CAT EDIT	

Name, Key or Label	Description Type, Keys	Page
EDITΣ	Copies statistical data in \(\sumeq DAT\) to MatrixWriter application.  O \(\begin{array}{c} \STAT\) \(\text{EDITZ}\)	368
EEX	Types E or moves cursor to existing exponent in command line.  O (EEX)	47
ELEC	Selects UNITS ELEC (electrical) menu.  O	
erg	Erg, energy (.0000001 kg·m²/s²) U 4- UNITS p.2 ENRG ERG	
ELSE	Begins ELSE clause. C PRG BRCH p.3 ELSE	496
END	Ends program structures.  C PRG BRCH p.2 END	494
ENG	Sets display mode to Engineering.  C  MODES ENG	58
ENRG	Selects UNITS ENRG (energy) menu. O	
ENTER	Enters contents of command line. If no command line is present, executes DUP.  O [ENTER]	99

Name, Key or Label	Description Type, Keys	Page
(ENTRY)	Switches Algebraic- and Program-entry modes.  O ENTRY	77
(EQUATION)	Selects EquationWriter application.  O ( EQUATION)	
EQ+	Adds selected equation to list in EQ.	272
€9 E@+	Removes the last entry from the list in EQ.  ———————————————————————————————————	272
EQ→	Separates equation into left and right sides.  C PRG OBJ EQ	91
ERASE	Erases PICT PLOTE ERASE C PPLOT ERASE	292

Name, Key or Label	Description Type, Keys	Page
ERRM	Returns last error message.  C PRG CTRL p.3 ERRM	542
ERRN	Returns last error number. C PRG CTRL p.3 ERRN	542
ERR0	Clears last error number. C PRG CTRL p.3 ERRØ	542
eV	Electron volt, energy (1.60219 × 10 <sup>-19</sup> kg·m²/s²) U — UNITS p.2 ENRG p.2 EV	
EVAL	Evaluates object. C [EVAL]	98
EXEC	Sets alarm execution action.  O  TIME	444
EXE(S)	Shows alarm-execution action.  (+) TIME   CAT   EXECS    O   TIME   EXECS	450

Name, Key or Label	Description Type, Keys	Page
EXII	Exits Selection environment.  O ( (EQUATION) ( EXIT	399
	Exits FCN (function) menu. O FCN EXIT	308
	Exits ZOOM menu. O ZOOM EXIT	305
EXP	Constant e raised to power of object in level 1.  A •••••	137
EXPAN	Expands algebraic object.  C	396
EXPFIT	Sets curve-fitting model to exponential.  C  STAT p.4 MODL EXPR	377
EXPM	Natural exponential minus 1 (e <sup>x</sup> - 1).  A MTH HYP p.2 EXPM	137
EXPR	Highlights subexpression for which specified object is top level function.	247
	O (SEQUATION) (SEXPR	398
EXPR=	Returns expression value or equation values.  O SOLVR EXPR=	265
EXTR	Moves graphics cursor to nearest extremum, displays coordinates, and returns them to stack.  O FCN EXTR	308
E^	Replace power-product with power-of-power.  O SEQUATION RULES	408

Name, Key or Label	Description Type, Keys	Page
E()	Replace power-of-power with power-product.  O	408
F	Farad, capacitance (1 A <sup>2</sup> ·s <sup>4</sup> /kg·m <sup>2</sup> ).  U ← UNITS p.2 ELEC F	
°F	Degrees Fahrenheit, temperature.  U — UNITS p.2 TEMP F	
FAST	Switches displaying equation names only and names plus contents of equations.  O CRT p.2 FRST	260
fath	Fathom, length (1.82880365761 m). U — UNITS LENG p.3 FATH	~
fbm	Board foot, volume (.002359737216 m³).  U — UNITS VOL p.4 FBM	
fc	Footcandle, illuminance (.856564774909 cd/m²) U 🔄 UNITS p.3 LIGHT FC	
FCN	Selects GRAPHICS FCN (function) menu DRRW FCN RUTO FCN O GRAPH FCN	

Name, Key or Label	Description Type, Keys	Page
FC?	Tests if specified flag is clear.  PRG TEST p.3 FC?  C MODES p.3 FC?	516
FC?C	Tests if specified flag is clear, then clears it.  PRG TEST p.3 FC?C  C MODES p.3 FC?C	516
Fdy	Faraday, electric charge (96487 A·s).  U	
fermi	Fermi, length (1 × 10 <sup>-15</sup> m).  U — UNITS LENG p.4 FERMI	
FINDALARM	Returns first alarm due after specified time.  C	454
FINISH	Terminates Kermit server mode.  C	615
FIX	Selects Fix display mode.  C  MODES FIX	58
flam	Footlambert, luminance (3.42625909964 cd/m²)  U  UNITS p.3 LIGHT FLAM	

Name, Key or Label	Description Type, Keys	Page
FLOOR	Next smaller integer.  F MTH PARTS p.3 FLOOR	148
FM,	Switches period and comma fraction mark.  O	58
FOR	Begins definite loop. C PRG BRCH FOR	506
	Types FOR NEXT.  O PRG BRCH  FOR	506
₽ FOR	Types FOR STEP. O PRG BRCH P FOR	508
M01442	Selects UNITS FORCE menu. O	
FP	Returns fractional part of a number.  F MTH PARTS p.3 FP	148
FREE	Replaces object in RAM with new copy of object.  C	649
FREEZE	Freezes one or more of three display areas.  C PRG DSPL p.4 FREEZ	344 523

Name, Key or Label	Description Type, Keys	Page
FS?	Tests if specified flag is set.  PRG TEST p.3 FS?  C MODES p.3 FS?	516
FS?C	Tests if specified flag is set, then clears it.  PRG TEST p.3 FS7C  C MODES p.3 FS7C	516
ft	International foot, length (.3048 m).  U	
ft^2	Square foot, area (.09290304 m²).  U ← UNITS BREA FT^2	
ft^3	Cubic foot, volume (.028316846592 m³).  U ———————————————————————————————————	
ftUS	Survey foot, length (.304800609601 m).  U	
ft/s	Feet/second, speed (.3048 m/s). U — UNITS SPEED FTZS	
ft*lbf	Foot-poundf, energy (1.35581794833 kg·m²/s²).  U — UNITS p.2 ENRG FT*LB	

Name, Key or Label	Description Type, Keys	Page
FUNCTION	Selects FUNCTION plot type. C PTYPE FUNC	327
F(X)	Displays value of function at x-value specified by cursor. Returns function value to stack.  O FCN p.2 F(X)	308
F'	Plots first derivative of function, replots function, and adds derivative to EQ.  O FCN p.2 F	309
g	Gram, mass (.001 kg). U ← UNITS MASS G	33/
ga	Standard freefall, acceleration (9.80665 m/s²).  U ← UNITS SPEED p.2 GR	20
gal	US gallon, volume (.003785411784 m³). U	
galC	Canadian gallon, volume (.00454609m³).  U ← UNITS ♥ OL p.2 GALC	
galUK	UK gallon, volume (.004546092 m³). U ﴿ UNITS VOL p.2 GALU	440
GET	Gets element from array or list.  C PRG OBJ p.4 GET	91

Name, Key or Label	Description Type, Keys	Page
GETI	Gets element from array or list and increments index.  C [PRG] [DBJ] [p.4] [GETI]	92
gf	Gram-force (.00980665 kg·m/s²). U — UNITS p.2 FORCE GF	
GOR	Superposes graphics object onto graphics object.  C PRG DSPL p.3 GOR	343
G0+	Sets top-to-bottom entry mode.  O ► MATRIX GO+	357
GO→	Sets left-to-right entry mode.  O ► MATRIX G0→	351
GRAD	Selects Grads mode.  C  MODES p.3 GRAD	139
grad	Grade, plane angle (.0025). U ← UNITS p.3 ANGL GRAD	
grain	Grain, mass (.00006479891 kg). U — UNITS MASS p.2 GRAIN	2 1 200 10 10 10
GRAPH	Enters Graphics environment.  C	301
(GRAPH)	Invokes scrolling mode.  GEQUATION GERAPH  DRAW GERAPH  HUTD GERAPH  GERAPH GERAPH	229 303

Name, Key or Label	Description Type, Keys	Page
→GROB	Converts object into graphics object.  C PRG DSPL p.3 → GRO	342
GXOR	Superposes inverting graphics object onto graphics object.  C PRG DSPL p.3 GXOR	343
Gy	Gray, absorbed dose (1 m²/s²). U ← UNITS p.3 RAD GY	
h	Hour, time (3600 s). U ← UNITS TIME H	
Н	Henry, inductance (1 kg·m²/A²·s²).  U	
*H	Adjusts vertical plot scale.  C PLOTR p.3 *H	319
ha	Hectare, area (10000 m²). U ← UNITS AREA p.2 HA	,
HALT	Halts program execution. C PRG CTRL HALT	484 523
HEX	Sets hexadecimal base.  MTH BASE HEX  C MODES p.4 HEX	208

Name, Key or Label	Description Type, Keys	Page
HISTPLOT	Draws histogram of data in ΣDAT.  C STAT p.3 HISTP	378
HISTOGRAM	Selects HISTOGRAM plot type.  C PTYPE p.2 HIST	328
HMS+	Adds in HMS format.	142
	C TIME p.3 HMS+	457
HMS-	Subtracts in HMS format.	142
	C TIME p.3 HMS-	457
HMS→	Converts from HMS to decimal format.	142
	C ← TIME p.3 HMS→	456
→HMS	Converts base 10 number to HMS format.	142
100:- 1	C ←TIME p.3 →HMS	456
HOME	Selects HOME directory.  C → HOME	122
HOUR	Sets alarm repeat interval to <i>n</i> hours.  O  HIME  ALRM  RPT  HOUR	445
hp	Horsepower, power (745.699871582 kg·m²/s³).  U ← UNITS p.2 POWR HP	
HR+	Increments time by one hour.  O  TIME	443
HR−	Decrements time by one hour.  O	443

Name, Key or Label	Description Type, Keys	Page
HYP	Selects MTH HYP (math hyperbolic) menu.  O MTH HYP	
Hz	Hertz, frequency (1/s). U — UNITS TIME HZ	- 12-7
İ	Symbolic constant i.  F @	144
IDN	Creates identity matrix of specified size.  C MTH MATR IDN	360
IF	Begins test clause.  C PRG BRCH IF	494
<b>4</b>	Types IF THEN END.  O PRG BRCH  IF	494
[→] IF	Types IF THEN ELSE END.  O PRG BRCH - IF	496

Name, Key or Label	Description Type, Keys	Page
IFERR	Begins test clause.	543
	C PRG BRCH p.3 IFERR	
<b>(5)</b>	Types IFERR THEN END.	543
	O PRG BRCH p.3 1FERR	
	Types IFERR THEN ELSE END.	545
	O PRG BRCH p.3 PIFERR	
IFT	IF-THEN command.	500
	C PRG BRCH p.3 IFT	
IFTE	IF-THEN-ELSE function.	500
	F PRG BRCH p.3 IFTE	
IM	Returns imaginary part of complex number	166
	or array.	
	F MTH PARTS IM	
in	Inch, length (.0254 m).	
As _	U SUNITS LENG IN	
in^2	Square inch, area (.00064516 m²).	
	U SUNITS AREA IN-2	
in^3	Cubic inch, volume (.000016387064 m <sup>3</sup> ).	
	U MUNITS VOL INAS	

Name, Key or Label	Description Type, Keys	Page
INCR	Increments value of specified variable.  C MEMORY INCR	513
INDEP	Specifies independent variable in a plot PLOTR INDEP C PLOT INDEP Recalls independent variable to stack PLOTR PINDEP O PLOT PINDEP	294 293
inHg	Inches of mercury, pressure (3386.38815789 kg/m·s²).  U — UNITS p.2 PRESS p.2 INHG	
inH2O	Inches of water, pressure (248.84 kg/m·s²).  U — UNITS p.2 PRESS p.2 INH20	
INPUT	Suspends program execution, displays message, and waits for data.  C PRG CTRL p.2 INPUT	524
INS	Switches between insert/replace character.  O	68

Name, Key or Label	Description Type, Keys	Page
INV	Reciprocal. A 1/x	61
IP	Integer part of real number.  F MTH PRRTS p.3 IP	148
T.R.ZIA	Switches IR and Wire transmission modes.  O     I/O SETUR IR/W	617
163 <b>3-0</b> 1	Moves graphics cursor to closest intersection in two-function plot, displays intersection coordinates, and returns coordinates to stack.  O FEN ISECT	308
ISOL	Isolates variable on one side of equation.  C	389
<b>(1)</b>	Selects I/O (input/output) menu.  O	624
J	Joule, energy (1 kg·m²/s²).  U — UNITS p.2 ENRG J	
K	Kelvins, temperature (1 K). U ← UNITS p.2 TEMP K	

Name, Key or Label	Description Type, Keys	Page
kcal	Kilocalorie, energy (4186 kg·m²/s²) U	
(4 <b>=1=</b>	Clears all levels above current level.  O +STK p.2 KEEP	72
KERRM	Returns text of most recently-received KERMIT error packet.  C ¶[/O] p.2 KERR	615
KEY	Returns number indicating last key pressed.  C PRG CTRL p.2 KEY	540
KEYS	Removes menu labels DRRW p.3 KEYS RUTO p.3 KEYS O	302
kg	Kilogram, mass (1 kg). U 🔄 UNITS MASS KG	
KGET	Gets data from another device.  C ← I/O KGET	615
KILL	Aborts all suspended programs.  C PRG CTRL KILL	484

Name, Key or Label	Description Type, Keys	Page
kip	Kilopound-force (4448.22161526 kg·m/s²).  U	
km	Kilometer, length (1 km). U	
km^2	Square kilometer, area (1 km²).  U ← UNITS AREA p.2 KM^2	
knot	Nautical miles per hour, speed (.514444444444 m/s).  U ← UNITS SPEED KNOT	
kph	Kilometers per hour, speed (.27777777778 m/s). U ← UNITS SPEED KPH	
1	Liter, volume (.001 m³).  U — UNITS VOL p.2 L	
LABEL	Labels axes with variable names and ranges PLOTR p.3 LABEL C PLOT p.3 LABEL	320

Name, Key or Label	Description Type, Keys	Page
LABEL	Labels axes with variable names and ranges DRAW LABEL RUTO LABEL O GRAPH LABEL	302
lam	Lambert, luminance (3183.09886184 cd/m²). U ← UNITS p.3 LIGHT p.2 LRM	
LAST	Returns previous argument(s) to stack.  C Must be keyed in.	
LASTARG	Returns previous argument(s) to stack.  C	64
LAST CMD	Displays previous contents of command line.  O ( LAST CMD)	77
[LAST MENU]	Selects last displayed page of previous menu.  O PLAST MENU	57
[LAST STACK]	Restores previous stack.  O	74

Name, Key or Label	Description Type, Keys	Page
lb	Avoirdupois pound, mass (.45359237 kg).  U	5.23
lbf	Pound-force (4.44822161526 kg·m/s²).  U ← UNITS p.2 FORCE LBF	
lbt	Troy pound, mass (.3732417 kg). U ← UNITS MASS LBT	12
LCD→	Returns graphics object to stack representing stack display.  C [PRG] DSPL p.4 LED→	344
→LCD	Displays specified graphics object in stack display.  C PRG DSPL p.4 → LCD	343
LENG	Selects UNITS LENG (length) menu.  O	TEAJI
	Enters current level number into level 1. O +STK p.2 LEVEL	72
[LIBRARY]	Selects LIBRARY menu.  O ← LIBRARY	

Name, Key or Label	Description Type, Keys	Page
LIBS	Lists all libraries attached to current directory.  C  MEMORY p.2 LIBS	653
	Selects UNITS LIGHT menu. O (4)(UNITS) p.3 LIGHT	
LINE	Draws line between coordinates in levels 1 and 2.  C PRG DSPL LINE	339
LINE	Draws line from mark to cursor DRRW p.2 LINE AUTO p.2 LINE O SGRAPH p.2 LINE	337
ΣLINE	Returns best-fit line for data in \$\times DAT\$ according to selected statistical model.  C \( \begin{align*} \text{STAT} \text{ p.3 } \text{ZLINE} \end{align*}	376
LINFIT	Sets curve-fitting model to linear.  C  STAT p.4 MODL LIN	377
LIST→	Returns list elements to stack.  C Must be typed in.	
→LIST	Combines specified objects into list.  C PRG 0BJ *LIST	92

Name, Key or Label	Description Type, Keys	Page
⇒LIST	Combines objects from level 1 to current level into a list.  O *STK *LIST	71
lm	Lumen, luminous flux (7.95774715459 × 10 <sup>-2</sup> cd). U	
LN	Natural (base e) logarithm.  A → LN	61
LNP1	Natural logarithm of (argument + 1).  A MTH HYP p.2 LNP1	138
LOG	Common (base 10) logarithm. A → LOG	137
LOGFIT	Set curve-fitting model to logarithmic.  C  STAT p.4 MODL LOG	377
LR	Calculates linear regression.  C	376
lx	Lux, illuminance (7.95774715459 × 10 <sup>-2</sup> cd/m²). U <b>(4) (UNITS)</b> p.3 LIGHT LX	
lyr	Light year, length (9.46052840488 × 10 <sup>15</sup> m).  U	
L*	Replace log-of-power with product-of-log.  O	408
LO	Replace product-of-log with log-of-power.  O 「TEQUATION」 RULES L()	408

Name, Key or Label	Description Type, Keys	Page
<¥	Merge-factors-left. O ← EQUATION ■ RULES ← M	406
M+ (4)	Executes +M until no change in subexpression.  O   RULES	410
M- <b>→</b>	Merge-factors-right. O ← EQUATION ■ RULES M→	407
<b>→</b> M→	Executes M+ until no change in subexpression.  O — EQUATION RULES  M+	410
m	Meter, length (1 m). U ← UNITS LENG M	
m^2	Square meter, area (1 m²). U ← UNITS AREA M^2	- 22
m^3	Cubic meter (Stere), volume (1 m³).  U	
MANT	Mantissa (decimal part) of number.  F MTH PARTS p.3 MANT	148

Name, Key or Label	Description Type, Keys	Page
∰¥f#T#K	Sets mark at cursor position.	302
	DRAW p.3 MARK AUTO p.3 MARK O    GRAPH p.3 MARK	17-
WASS	Selects UNITS MASS menu. O	
↑MATCH	Match-and-replace, beginning with subexpressions.  C ← ALGEBRA p.2 ★MAT	415
ĮMATCH	Match-and-replace, beginning with top- level expression.  C ← ALGEBRA p.2 → MRT	415
VIETHE	Selects MTH MATR (math matrices) menu.  O MTH MATR	-
MATRIX	Selects MatrixWriter application.  O ► [MATRIX]	
MAX	Maximum of two real numbers.  F MTH PARTS p.2 MAX	148

Name, Key or Label	Description Type, Keys	Page
MAXR	Maximum machine-representable real number (9.99999999999999999999999999999999999	144
ΜΑΧΣ	Maximum column values in statistics matrix in ∑DAT.  C ← STAT p.2 MAXZ	374
MEAN	Calculates mean of statistical data in ∑DAT.  C ← STAT p.2 MEAN	374
MEM ()	Bytes of available memory.  C  MEMORY MEM	101
MEMORY MEMORY	Selects MEMORY menu.  O  MEMORY  Selects MEMORY Arithmetic menu.	Œ.
	O (MEMORY)	(r
MENU	Displays built-in or custom menu.	
	MODES MENU	213
	C PRG ETRL p.2 MENU	534

Name, Key or Label	Description Type, Keys	Page
MERGE	Merges plug-in RAM card memory with main memory.  C MEMORY p.3 MERG	643
μ	Micron, length (1 × 10 <sup>-6</sup> m).  U ← UNITS LENG p.4	
MeV	Mega electron volt, energy (1.60219 × 10 <sup>-13</sup> kg·m²/s²).  U	370
mho	Mho, electric conductance (1 A <sup>2</sup> ·s <sup>3</sup> /kg·m <sup>2</sup> ).  U — UNITS p.2 ELEC p.2 MHO	-14
mi	International mile, length (1609.344 m).  U	
mi^2	International square mile, area (2589988.11034 m²).  U	
mil	Mil, length (.0000254 m). U — UNITS LENG p.4 MIL	
min	Minute, time (60 s). U	
MIN	Minimum of two real numbers.  F MTH PARTS p.2 MIN	148

Name, Key or Label	Description Type, Keys	Page
¥##Z	Sets alarm repeat interval in minutes.  O	445
MINR	Minimum machine-representable real number (1.00000000000E <sup>-499</sup> ).  F MTH PARTS p.4 MINR	144
	Increments system time by one minute.  O	443
Water	Decrements system time by one minute.  O	443
ΜΙΝΣ	Finds minimum column values in statistics matrix in ΣDAT.  C STAT p.2 MINΣ	374
miUS	US statute mile, length (1609.34721869 m). U	
miUS^2	US statute square mile, area (258998.47032 m²).  U	
mm	Millimeter, length (.001 m). U ← UNITS LENG MM	
mmHg	Millimeter of mercury (torr), pressure (133.322368421 kg/m·s²).  U ← UNITS p.2 PRESS MMH	

Name, Key or Label	Description Type, Keys	Page
ml	Milliliter (cubic centimeter), volume (1 × 10 <sup>-6</sup> m³).	
	U SUNITS VOL p.3 ML	
WIL.	Switches multi-line and single-line display.  O MODES p.2 ML	221
MOD	Modulo. F MTH PARTS p.2 MOD	148
MODES	Selects MODES menu.	215
	O MODES	
<b>MODES</b>	Selects MODES Customization menu.	
	O MODES	
MODL	Selects STAT MODL (statistics model) menu.  O STAT p.4 MODL	377
mol	Mole, mass (1 mol).	1
	U (1) UNITS MASS p.3 MOL	
Мрс	Megaparsec, length	
	(3.08567818585 × 10 <sup>22</sup> m).	11
	U SUNITS LENG p.2 MPC	
mph	Miles per hour, speed (.44704 m/s).  U	

Name, Key or Label	Description Type, Keys	Page
(MTH)	Selects MTH (math) menu. O MTH	
ŭ/Z»	Switches date display format.  O  TIME SET M/D	442
m/s	Meters per second, speed (1 m/s).  U ← UNITS SPEED M/S	
N	Newton, force (1 kg·m/s²). U ← UNITS p.2 FORCE N	
ΝΣ	Returns number of rows in ∑DAT.  C ← STAT p.5 N∑	383
NEG	Negate. A +/-	134
HEW	Takes algebraic or matrix from stack, prompts for name, stores named algebraic in $EQ$ , or named matrix in $\Sigma DAT$ .	
	SOLVE NEW  O STAT NEW	257 368
NEWOB	Decouples object from list or variable name.  C  MEMORY p.2 NEWO	
NEXT	Ends a definite-loop structure.  C PRG BRCH p.2 NEXT	502 506

Name, Key or Label	Description Type, Keys	Page
NEXT	Displays but does not execute next one or two objects in suspended program.  O PRG CTRL NEXT	484
nmi	Nautical mile, length (1852 m).  U ← UNITS LENG p.3 NMI	
NONE	Cancels alarm repeat interval and returns to TIME ALRM menu.  O  TIME ALRM RPT NONE	445
NOT	Logical or binary NOT.  PRG TEST NOT  F MTH BASE p.4 NOT	493 210
NUM	Returns character code of first character in string.  C PRG OBJ p.3 NUM	92
→NUM	Evaluates algebraic to number.  C → NUM	127
%19%4 <b>=</b> €	Rotates list of equations in EQ SOLVR NXEQ O FCN p.2 NXEQ	272 309
(NXT)	Selects next page of menu.  O NXT	56

Name, Key or Label	Description Type, Keys	Page
9)5(0)	Selects PRG OBJ (program object) menu. O PRG OBJ	
OBJ→	Returns object components to stack.  C PRG 0BJ 0BJ+	93
ОСТ	Sets octal base.  MTH BASE OCT  C  MODES p.4 OCT	208
OFF	Turns calculator off.  O (P)(OFF)	25
OFF	Turns calculator off. C PRG CTRL p.3 OFF	540
OLDPRT	Remaps HP 48 character set to match HP 82240A Infrared Printer.  C PRINT p.2 DEDPR	603
ON	Turns calculator on. O ON	25

Name, Key or Label	Description Type, Keys	Page
OPENIO	Opens serial port. C	615
OR	Logical or binary OR.  MTH BASE p.4 OR  F PRG TEST OR	210 493
ORDER	Rearranges VAR menu in order specified in list.  C	113
ORDER	Puts selected equation at top of Equation Catalog list.  ———————————————————————————————————	260
	O ALGEBRA p.2 ORDER  Puts selected statistical data at top of Statistics Catalog list.  O STAT CRT p.2 ORDER	372
OVER	Duplicates object in level 2 in level 1.  C [PRG] STK OVER	79
OZ	Ounce, mass (.028349523135 kg). U	

Name, Key or Label	Description Type, Keys	Page
ozfl	US fluid ounce, volume (2.95735295625 × 10 <sup>-5</sup> m³). U ← UNITS VOL p.3 0ZFL	
ozt	Troy ounce, mass (.031103475 kg).  U — UNITS MASS p.2 0ZT	
ozUK	UK fluid ounce, volume (2.8413075 × 10 <sup>-5</sup> m³). U — UNITS VOL p.3 OZUK	
Р	Poise, dynamic viscosity (.1 kg/m·s) U	
Pa	Pascal, pressure (1 kg/m·s²) U ← UNITS p.2 PRESS PA	
PARAMETRIC	Selects PARAMETRIC plot type. C PTYPE PARA	327
PARITY	Selects one of 5 possible parity settings.  C	617
PARTS	Selects MTH PARTS menu. O MTH PARTS	
PATH	Returns list containing path to current directory.  C  MEMORY PATH	120

Name, Key or Label	Description Type, Keys	Page
рс	Parsec, length (3.08567818585 × 10 <sup>16</sup> m).  U	
PDIM	Changes size of PICT PLOTR p.3 PDIM C → PLOT p.3 PDIM	325
→ PDIM	Recalls size of PICT to stack PLOTR p.3 PDIM O PLOT p.3 PDIM	319
pdl	Poundal, force (.138254954376 kg·m/s²).  U	
PERM	Permutations. F MTH PROB PERM	147
PGDIR	Purges specified directory.  C	123
ph	Phot, illuminance (795.774715459 cd/m²) U	89

Name, Key or Label	Description Type, Keys	Page
PICK	Copies object in level <i>n</i> to level 1.  C PRG STK PICK	79
23(4)	Copies object in current level to level 1.  O +STK PICK	71
PICT	Returns PICT to level 1. C PRG DSPL PICT	341
PIXOFF	Turns off specified pixel in PICT.  C PRG DSPL p.2 PIXOF	339
PIXON	Turns on specified pixel in PICT.  C PRG DSPL p.2 PIXON	339
PIX?	Tests whether specified pixel in PICT is on or off.  C PRG DSPL p.2 PIX?	339
pk	Peck, volume (.0088097675 m³).  U — UNITS VOL p.4 PK	
PKT	Sends KERMIT commands to a server.  C	615

Name, Key or Label	Description Type, Keys	3 M	Page
PLOT	Selects PLOT menu. O		, A
	Selects PLOT PLOTR menu.  O PLOT		
PLOT	Makes the selected entry the current statistical matrix and displays the third page of the STAT menu.		371
PLOTR	O STAT CAT PLOT Selects PLOT PLOTR menu.	-	
14-011	SPLOT PLOTE  SPLOT CAT PLOTE	uli;	6.15
	O SOLVE CAT PLOTE		
PMAX	Sets upper-right plot coordinates.  C Must be typed in.		G.
PMIN	Sets lower-left plot coordinates.  C Must be typed in.		
POLAR	Switches rectangular and polar coordinates.  O POLAR		158
POLAR	Selects POLAR plot type. C PTYPE POLAR		327

Name, Key or Label	Description Type, Keys	Page
POS	Returns the position of substring in string or object in list.  C PRG OBU p.3 POS	93
zent.	Selects UNITS POWR (power) menu.  O	
PREDV	Predicted value. C Must be typed in.	
PREDX	Returns predicted value for independent variable, given value of dependent variable.  C	376
PREDY	Returns predicted value for dependent variable, given value of independent variable.  C  STAT p.4 PREDY	377
PRESS	Selects UNITS PRESS (pressure) menu.  O	
← PREV	Selects previous page of menu.  O	56
<b>PREV</b>	Selects first page of menu.  O → PREV	56

Name, Key or Label	Description Type, Keys	Page
PRG	Selects PRG (program) menu. O [PRG]	
(PRINT)	Selects PRINT menu. O 「PRINT	
PRLCD	Prints display.  C  PRINT PRICD  O Simultaneously press ON MTH	603
PROB	Selects MATH PROB (probability) menu.  O MTH PROB	
PROMPT	Displays prompt string in status area and halts program execution.  C PRG CTRL p.2 PROM	521
PRST	Prints all objects on stack.  C  PRINT PRST	603
PRSTC	Prints all objects on stack in compact format.  C   PRINT PRSTC	603

Name, Key or Label	Description Type, Keys	Page
PRVAR	Prints name and contents of one or more variables (including port names).  C  PRINT PRYSE	603
PR1	Prints object in level 1.	603
psi	Pounds per square inch, pressure (6894.75729317 kg/m·s²).  U	
pt	Pint, volume (.000473176473 m³).  U ← UNITS VOL p.2 PT	
PTYPE	Selects PLOT PTYPE menu.  ( PLOT PTYPE PLOTR p.2 PTYPE O PLOT p.2 PTYPE	
PURGE	Purges one or more specified variables.  C — [PURGE]	114
PURGE	Purges one or more specified variables. If only one untagged variable specified, saves previous contents for recovery by LASTARG.  O  PURGE	114

Name, Key or Label	Description Type, Keys	Page
PURG	Purges selected equation.  SOLVE CAT p.2 PURG  PLOT CAT p.2 PURG	260
	O ALGEBRA CAT p.2 PURG Purges selected statistical matrix. O STAT CAT p.2 PURG	372
	Purges selected alarm.  TIME CAT PURG  TIME PURG	450
PUT	Replaces element in array or list.  C [PRG] OBJ p.4 PUT	94
PUTI	Replaces element in array or list and increments index.  C [PRG] [08J] p.4 [PUTI]	94
PVARS	Returns list of current backup objects and libraries within a port.  C MEMORY p.2 PYARS	647
PVIEW	Displays PICT with specified pixel at upper-left corner of display.  C [PRG] DSPL PYIEW	342

Name, Key or Label	Description Type, Keys	Page
PWRFIT	Set curve-fitting model to Power.  C	377
PX→C	Converts pixel coordinates to user-unit coordinates.  C PRG DSPL p.2 PX+C	324
→Q	Converts number to fractional equivalent.  C ← → □	134
QUAD	Finds solutions of first or second order polynomial.  C [A][ALGEBRA] QUAD	391
QUOTE	Returns argument expression unevaluated.  F (A)[ALGEBRA] p.2 QUOT	
qt	Quart, volume (.000946352946 m³). U ← UNITS VOL p.2 QT	- :
→Qπ	Calculates and compares quotients of number and number/π.  C ← ALGEBRA p.2 → Qπ	134
r	Radian, plane angle (.1591549343092).  U 🕣 [UNITS] p.3 RNGL R	
R	Roentgen, radiation exposure (.000258 A·s/kg).  U   U   RRD p.2 R	

Name, Key or Label	Description Type, Keys	Page
°R	Degrees Rankine, temperature.  U	
rad	Rad, absorbed dose (.01 m²/s²).  U	÷
RAD	Sets Radians mode.  C  MODES p.3 RAD	139
RAD	Switches Radians and Degrees mode.  O ( RAD)	30
2(1)	Selects UNITS RAD (radiation) menu.  O  RAD	
RAND	Returns random number. C MTH PROB RAND	147
RATIO	Prefix form of / used by EquationWriter application.  F Must be typed in.	

Name, Key or Label	Description Type, Keys	Page
RCEQ	Returns equation in EQ to level 1.  PLOT STER  SOLVE STER  PLOTR DRAW  C PPLOT DRAW	256
RCL	Recalls object stored in specified variable to stack.  C	110
RCL	Inserts algebraic from level 1 into EquationWriter equation.  O PRCL	246
RCLALARM	Recalls specified alarm from system alarm list.  C	453
RCLF	Returns binary integer representing states of system flags.  C MODES p.2 RCLF	518
RCLKEYS	Returns list of current user-key assignments.  C MODES ROLK	220
RCLMENU	Returns menu number of current menu.  C MODES p.2 RCLM	535

Name, Key or Label	Description Type, Keys	Page
RCLS	Recalls current statistical matrix in \$\sum_DAT\$.  C \( \bigs\) STO\$	368
RCWS	Recalls binary integer wordsize.  C MTH BASE RCUS	207
rd	Rod, length (5.0292100584 m). U — UNITS LENG p.3 RD	
RDM	Redimensions array. C MTH MATR RDM	360
RDZ	Sets random number seed. C MTH PROB RDZ	147
RE	Returns real part of complex number or array.  F MTH PARTS RE	166
RECN	Waits for stack-specified data from remote source running Kermit software.  C ← [7] p.2 RECN	615
RECV	Waits for sender-specified data from remote source running Kermit software.  C	614

Name, Key or Label	Description Type, Keys	Page
rem	Rem, dose equivalent (.01 m²/s²).  U	
REPEAT	Begins REPEAT clause. C PRG BRCH p.2 REPER	512
REPL	Replaces portion of object with another like object.	
	PRG OBJ p.3 REPL C PRG DSPL p.3 REPL	95 343
REPL	Replaces portion of PICT with level-1 graphics object DRAW p.3 REPL AUTO p.3 REPL	341
REPL	O GAGRAPH p.3 REPL  Replaces specified subexpression with algebraic from stack.  O GAGRAPH p.3 REPL	247

Name, Key or Label	Description Type, Keys	Page
RES	Sets spacing between plotted points PLOTR p.2 RES	321
RES	C PLOT p.2 RES  Recalls spacing to stack.  PLOTR p.2 PRES  O PLOT p.2 RES	318
RESET	Resets plot parameters in PPAR in the current directory to their default states and erases and resizes PICT PLOTR p.2 RESET  O PLOT p.2 RESET	323
RESTORE	Replaces HOME directory with backup copy.  C  MEMORY p.3 RESTO	648

Name, Key or Label	Description Type, Keys	Page
REVIEW	Displays statistical data in ΣDAT.	Same
	O STAT REVIEW	368
	Displays current equation and plot parameters.	IR
	PLOTE ( REVIEW)	294
	PLOT ( REVIEW	1111
	DRAW (STREVIEW)	- 66-
	PUTO (STREVIEW)	10000
	O GRAPH REVIEW	303
	Displays current equation.	
	O SOLVE REVIEW	256
	← PLOT ← REVIEW	290
	Displays current equation and values of SOLVR variables.	265
	O SOLVR ( REVIEW)	2011
	Displays unit names corresponding to selected menu.	191
	O ( UNITS ( REVIEW)	
	Displays pending alarm.	439
	O TIME REVIEW	1
	In other menus: Lists operation names and	112
	types.	
	O FREVIEW	

Name, Key or Label	Description Type, Keys	Page
RL	Rotates left by one bit. C MTH BRSE p.2 RL	211
RLB	Rotates left by one byte.  C MTH BASE p.2 RLB	211
RND	Rounds fractional part of number or name.  F MTH PARTS p.4 RND	148
RNRM	Calculates row norm of array.  C MTH MATR p.2 RNRM	360
ROLL	Moves object in level (n + 1) to level 1.  C PRG STK ROLL	79
	Rolls object in current level to level 1. O *STK ROLL	71
ROLLD	Moves object in level 2 to level n.  C PRG STK ROLLD	79
RULLD	Moves object in level 1 to current level.  O +STK_ROLLD	71
ROOT	Solves for unknown variable in equation.  C	256

Name, Key or Label	Description Type, Keys	Page
ROOT	Moves graphics cursor to intersection of function plot and x-axis, displays value of root, returns value to stack.  O ← GRAPH FCN ROOT	308
ROT	Moves object in level 3 to level 1.  C PRG STK ROT	79
27.(0)4	Inserts row of zeros at current row.  O ►MATRIX p.2 +ROW	351
=:30!	Deletes current row. O ►MATRIX p.2 -ROW	351
RPT	Selects TIME ALRM RPT (alarm repeat) menu.  O	
RR	Rotates right by one bit. C MTH BASE p.2 RR	211
RRB	Rotates right by one byte.  C MTH BASE p.2 RRB	211
RSD	Calculates correction to solution of system of equations.  C MTH MATR RSD	362
RULES	Activates RULES transformation menu for specified object.  O ← EQUATION ■ RULES	398

Name, Key or Label	Description Type, Keys	Page
R→B	Real-to-binary conversion.  C MTH BASE p.2 R+B	210
R→C	Real-to-complex conversion.  C PRG 08J pg.2 R+C	95
R→D	Radians-to-degrees conversion.  F MTH VECTR p.2 R+D	142
R&Z	Selects Polar/Cylindrical mode.  MTH VECTR R&Z  O  MODES p.3 R&Z	171
RAZ	Selects Polar/Spherical mode.  MTH VECTR R&&  O  MODES p.3 R&&	171
S	Second, time (1 s).  U — UNITS TIME S	
S	Siemens, electric conductance (1 A <sup>2</sup> ·s <sup>3</sup> /kg·m <sup>2</sup> ).  U — UNITS p.2 ELEC p.2 S	

Name, Key or Label	Description Type, Keys	Page
SAME	Tests two objects for equality.  C PRG TEST SAME	492
sb	Stilb, luminance (10000 cd/m²) U	
SBRK	Sends serial break. C	633
SCALE	Sets scale of PLOT axes PLOTR p.2 SCALE C PPLOT p.2 SCALE	295
SCALE	Recalls scale to stack PLOTR p.2 P SCALE O PLOT p.2 SCALE	294
SCATRPLOT	Draws scatter plot of statistical data in SDAT.  C STAT p.3 SCATE	379
SCATTER	Selects SCATTER plot type. C PTYPE p.2 SCATT	328

Name, Key or Label	Description Type, Keys	Page
SCI	Selects Scientific display mode.  C  MODES SCI	58
SCLS	Autoscales data in ΣDAT for scatter plot.  C Must be typed in.	
SCONJ	Conjugates contents of variable.  C MEMORY p.2 SCON	116
SDEV	Calculates standard deviation.  C  STAT p.2 SDEV	374
SEC	Sets alarm repeat interval to <i>n</i> seconds.  O	445
SEC+	Increments current time by 1 second.  O  TIME	443
SEC-	Decrements current time by 1 second.  O  TIME ADJST SEC-	443
SEND	Sends contents of variable to another device.  C	614
SERVER	Puts HP 48 into Kermit Server mode.  GIVO SERV  C PIVO	614

Name, Key or Label	Description Type, Keys	Page
Sin	Selects TIME SET menu.	
	O STIME SET	
	Sets alarm.	445
	O TIME ALRM SET	
	Selects I/O SETUP menu.	
	O SETUP	
SF	Sets specified flag.	516
	PRG TEST p.3 SF	
	C MODES p.2 SF	
SHOW	Reconstructs expression to resolve implicit	394
	variable name.	
	C (ALGEBRA) SHOW	
SIGN	Returns sign of number.	149
	F MTH PARTS SIGN	
SIN	Sine.	140
	A SIN	
SINH	Hyperbolic sine.	137
	A MTH HYP SINH	

Name, Key or Label	Description Type, Keys	Page
SINV	Replaces contents of variable with its inverse.  C MEMORY p.2 SINV	116
SIZE	Finds dimensions of list, array, string, algebraic object, or graphics object.  PRG OBJ p.3 SIZE  C PRG DSPU p.2 SIZE	95 342
+SKIP	Moves cursor left to next logical break.	68
SKIP>	Moves cursor right to next logical break.  • EDIT SKIP*  O EDIT SKIP*	68
SL	Shifts left by one bit. C MTH BASE p.3 SL	211

Name, Key or Label	Description Type, Keys	Page
SLB	Shifts left by one byte.  C MTH BRSE p.3 SLB	211
SLOPE	Calculates and displays slope of function at cursor position, returns slope to stack.  O FCN SLOPE	308
slug	Slug, mass (14.5939029372 kg). U	
SNEG	Negates contents of variable.  C ► MEMORY p.2 SNEG	116
SOLVE	Selects SOLVE menu. O SOLVE	
SOLVR	Selects SOLVR menu.  SOLVE SOLVR  SOLVE CAT SOLVR  SOLVE  PLOT CAT SOLVR  A PLOT CAT SOLVR  A PLOT SOLVR	
SPC	Types a blank space in command line.  O SPC	
SPEED	Selects UNITS SPEED menu.  O	

Name, Key or Label	Description Type, Keys	Page
SQ	Returns square of level-1 object.  A	134
SR	Shifts right by one bit. C MTH BRSE p.3 SR	211
sr	Steradian, solid angle (7.95774715459 × 10 <sup>-2</sup> ). U ← UNITS p.3 RNGL SR	
SRB	Shifts right by one byte.  C MTH BASE p.3 SRB	211
SRECV	Reads specified number of characters from I/O port.  C     [/O] p.3   SRECY	633
SST	Single-steps through suspended program.  O PRG CTRL SST	484
SST+	Single-steps through suspended program and its subroutines.  O PRG CTRL SST+	486
st	Stere, volume (1 m³).  U ———————————————————————————————————	
St	Stokes, kinematic viscosity (.0001 m²/s) U  ( UNITS p.3 VISC ST	

Name, Key or Label	Description Type, Keys	Page
START	Begins definite loop. C PRG BRCH START	502
<b>START</b>	Types START NEXT.  O PRO BROH START	502
<b>→</b> START	O PRG BRCH START Types START STEP. O PRG BRCH PSTART	504
STAT	Selects STAT (statistics) menu.  O	
STAT	Selects page 2 of STAT menu.  O FSTAT	
STD	Selects Standard display mode.  C ← MODES STD	58
STEP	Ends definite loop.	504
	C PRG BRCH p.2 STEP	508

Name, Key or Label	Description Type, Keys	Page
STEQ	Stores level 1 equation in EQ.  STER	257
	PLOTE STEQ  C SOLVE STEQ	
STIME	Sets serial transmit/receive timeout.  C	633
3 IK	Selects PRG STK (program stack) menu.  O PRG STK	
STK	Switches Last Stack recovery on and off.  O	221
+STK	Selects Interactive Stack.	
	EDIT +STK	70
	O MATRIX p.2 +STK	351

Name, Key or Label	Description Type, Keys	Page
≯STK	Copies selected equation to level 1 CRT p.2 +STK O [->][ALGEBRA] p.2 +STK	260
	Copies selected matrix to level 1.  O STAT CAT +STK	371
	Copies selected alarm to level 1.	450
	O TIME STK Copies selected matrix element to level 1. O MATRIX p.2 STK	351
STO	Stores object in variable.  C STO	107
(STO)	Stores object in variable and saves previous contents of variable for recovery by LASTARG.  O STO	107
STO	Returns EquationWriter equation or PICT to stack.	229
STOALARM	O STO  Stores level 1 alarm in system alarm list.  C  TIME ALRM p.2 STORL	303 453
STOF	Sets state of system and user flags.  C MODES p.2 STOF	518
STOKEYS	Makes multiple user-key assignments.  C → MODES STOK	217

Name, Key or Label	Description Type, Keys	Page
STO+	Adds specified number or array to contents of specified variable.  C MEMORY STO+	115
STO-	Subtracts specified number or array from contents of specified variable.  C MEMORY STO	116
STO*	Multiplies contents of specified variable by specified number.  C MEMORY STOX	116
STO/	Divides contents of specified variable by specified number.  C MEMORY STOZ	116
STOΣ	Stores current statistics matrix in \(\sum_DAT\).  C \(\begin{array}{c} \begin{array}{c} \b	368
STR→	Converts string to component objects.  C Must be typed in.	
→STR	Converts object into string.  C PRG 0BJ +STR	95
STWS	Sets binary integer wordsize.  C MTH BASE STWS	207
SUB	Extracts specified portion of list or string, or graphics object.	
119	PRG OBJ p.3 SUB C PRG DSPL p.3 SUB	96 343

Name, Key or Label	Description Type, Keys	Page
SUB	Returns specified portion of PICT to stack DRAW p.3 SUB	341
	O GRAPH p.3 SUB	
SUB	Returns specified subexpression to stack.  O	398
Sv	Sievert, dose equivalent (.01 m²/s²)  U — UNITS p.3 RAD SY	
SWAP	Exchanges objects in levels 1 and 2.  C	63
SYM	Switches Symbolic and Numerical Results mode.  O  MODES SYM	144
SYSEVAL	Evaluates system object. Use only as specified by HP applications.  C Must be typed in.	

Name, Key or Label	Description Type, Keys	Page
t	Metric ton, mass (1000 kg). U	
T	Tesla, magnetic flux (1 kg/As²).  U 🔄 UNITS p.2 ELEC p.2	
<b>CT</b>	Move term left.  O ← EQUATION ← RULES ← T	402
	Executes *T until no change in subexpression.  O * EQUATION   RULES   +T	410
<b>;</b>	Move term right.  O	402
	Executes T+ until no change in subexpression.	410
	O SEQUATION RULES	
%Т	Returns percent fraction that level-1 is of level-2.  F MTH PARTS p.2 %T	138
→TAG	Combines objects in levels 1 and 2 to create tagged object.  C PRG OBJ → TAG	96
TAN	Tangent. A TAN	140

Name, Key or Label	Description Type, Keys	Page
TANH	Hyperbolic tangent. A MTH HYP TANH	137
TAYLR	Calculates Taylor's polynomial.  C (1) [ALGEBRA] TAYLR	426
tbsp	Tablespoon, volume (1.47867647813 × 10 <sup>-5</sup> m³).  U	
TEMP	Selects UNITS TEMP (temperature) menu.  O	
TEST	Selects PRG TEST (program test) menu.  O PRG TEST	
TEXT	Displays stack display. C PRG DSPL p.4 TEXT	344
THEN	Begins THEN clause. C PRG BRCH p.2 THEN	494
therm	EEC therm, energy (105506000 kg·m²/s²) U	
TICKS	Returns system time as binary integer in units of clock ticks.  C  TIME p.2 TICKS	456

Name, Key or Label	Description Type, Keys	Page
TIME	Returns current time as a number.  C	456
TIME	Selects TIME menu.  O   TIME	
TIME	Selects Alarm Catalog.  O	449
	Selects UNITS TIME menu.  O ———————————————————————————————————	410
→TIME	Sets system time.  C ← TIME SET → TIM	442
) Bevi	Sets alarm time.  O	445
TLINE	Switches pixels on line defined by coordinates in levels 1 and 2.  C PRG DSPL TLINE	339

Name, Key or Label	Description Type, Keys	Page
TLINE	Switches pixels on and off on line between mark and cursor.	337
	DRAW p.2 TLINE	1900
	O GRAPH p.2 TLINE	
TMENU	Displays list-defined menu but does not change contents of CST.  C MODES p.2 TMEN	539
ton	Short ton, mass (907.18474 kg).  U — UNITS MASS p.2 TON	
tonUK	Long (UK) ton, mass (1016.0469088 kg). U ← UNITS MRSS p.2 TONU	
torr	Torr (mmHg), pressure (133.322368421 kg/ms²). U ← UNITS p.2 PRESS TORR	
TOT	Sums each column of matrix in ΣDAT.  C STAT p.2 TOT	374
TRANSIO	Selects one of three character translation settings.  C	618

Name, Key or Label	Description Type, Keys	Page
1426*	Expands trigonometric and hyperbolic functions of sums and differences.  O	409
TRN	Transposes matrix. C MTH MATR TRN	360
TRNC	Truncates (rounds down) number in level 2 as specified in level 1.  F MTH PARTS p.4 TRNC	149
TRUTH	Selects TRUTH plot type. C PTYPE TRUTH	327
tsp	Teaspoon, volume (4.92892159375 × 10 <sup>-6</sup> m³). U (UNITS) VOL p.3 TSP	
TSTR	Converts date and time in number form to string form.  C  TIME p.2 TSTR	455
TVARS	Returns variables containing specified object type.  C  MEMORY p.2 IVARS	98
TYPE	Returns type-number of argument object.  PRG OBJ p.2 TYPE  C PRG TEST TYPE	97
u	Unified atomic mass (1.66057 × 10 <sup>-27</sup> kg).  U	

Name, Key or Label	Description Type, Keys	Page
UBASE	Converts unit object to SI base units.  F DUNITS UBASE	196
UFACT	Factors specified compound unit.  C PUNITS UFACT	199
→UNIT	Combines objects in levels 1 and 2 to create unit object.  PRG OBJ p.2 *UNIT  C PUNITS *UNIT	96
(1) UNITS	Selects UNITS Catalog menu.  O  Selects UNITS Command menu.  O  UNITS	
UNTIL	Begins UNTIL clause. C PRG BRCH p.2 UNTIL	510
UPDIR	Makes parent directory the current directory.  C ← UP	122
USR	Turns User mode on and off. O 🔄 USR	216
UTPC	Returns probability that chi-square random variable is greater than x.  C MTH PROB p.2 UTPC	384

Name, Key or Label	Description Type, Keys	Page
UTPF	Returns probability that Snedecor's F random variable is greater than x.  C MTH PROB p.2 UTPF	384
UTPN	Returns probability that normal random variable is greater than x.  C MTH PROB p.2 UTPN	384
UTPT	Returns probability that Student's t random variable is greater than x.  C MTH PROB p.2 UTPT	384
UVAL	Returns scalar of specified unit object.  F	206
V	Volt, electrical potential (1 kg·m²/A·s³).  U	
VAR	Calculates variance of statistical data columns in ΣDAT.  C Must be typed in.	
VAR	Selects VAR (variables) menu.  O [VAR]	112
1-VAR	Makes the selected entry the current statistical matrix and displays the second page of the STAT menu.  O STAT CATE 1-VAR	371
2-VAR	Makes the selected entry the current statistical matrix and displays the fourth page of the STAT menu.  O STAT SCAT 2-VAR	371
VARS	Returns list of variables in current directory.  C	113

Name, Key or Label	Description Type, Keys	Page
VEC	Switches vector and array modes.  O MATRIX VEC	351
VECTR	Selects MTH VECTR (math vector) menu.  O MTH VECTR	9133
YIEW	Copies level 1 object into appropriate environment for viewing.	67
	O Topies object in current level into appropriate environment for viewing.	73
	O ↑STK VIEW Displays selected equation.	260
	O CAT VIEW Displays selected matrix. O ( STAT) CAT VIEW	372
	Displays selected alarm.  O	450
→ VIEW	Copies object stored in variable in the current level into appropriate	71
	environment for viewing. O ↑STK → VIEW	
VISC	Selects UNITS VISC (viscosity) menu.  O	
(VISIT)	If argument is name, copies contents of associated variable into command line for editing. If argument is a stack level number, copies object in that level into command line for editing.  O   VISIT	66

Name, Key or Label	Description Type, Keys	Page
VOL	Selects UNITS VOL (volume) menu. O	
VTYPE	Returns type number of object stored in local or global name.  C PRG OBU p.2 VTYPE	97
→V2	Combines two real numbers into a 2-D vector or complex number.  C MTH VECTR p.2 >V2	167
→V3	Combines three real numbers into 3-D vector.  C MTH VECTR p.2 +V3	183
V→	Separates 2- or 3-element vector according to current angle mode.  C MTH VECTR p.2 V+	167
W	Watt, power (1 kg·m²/s³)  ← UNITS p.2 POWR W  U ← UNITS p.2 ELEC W	
*W	Adjusts horizontal plot scale.  C PLOTR p.3 *W	319
WAIT	Halts program execution for specified number of seconds or until key pressed.  C PRG CTRL p.2 WAIT	534
Wb	Weber, magnetic flux (1 kg·m²/A·s²).  U ← UNITS p.2 ELEC p.2 WB	

Name, Key or Label	Description Type, Keys	Page
Mi=4=4K	Sets alarm repeat interval to n weeks.  O TIME BLRM RPT WEEK	445
WHILE	Begins indefinite loop.  C [PRG] BRCH WHILE	512
	Types WHILE REPEAT END O PRG BRCH HWHILE	512
WID-	Increases column width and decrements number of columns.  O MATRIX WID+	351
<#UND	Decreases column width and increments number of columns.  O MATRIX +WID	351
X	Selects x-axis zoom. O ZOOM X	305
ΣΧ	Returns sum of data in independent column in ΣDAT.  C STAT p.5 ΣΧ	383
ΣΧ^2	Returns sum of squares of data in independent column in ΣDAT.  C STAT p.5 ΣΧ^2	383
XAUMEO	Selects x-axis zoom with autoscaling.  O ZOOM XRUTO	305

Name, Key or Label	Description Type, Keys	Page
XCOL	Specifies independent-variable column in matrix in <i>SDAT</i> .	376
	C STAT p.3 XCOL	- 7
r XeoL	Recalls independent-variable column number to stack.	376
	O STAT p.3 PXCOL	
XMIT	Without Kermit protocol, performs serial send of string.  C	632
XOR		
XOR	Logical or binary exclusive OR.  MTH BRSE p.4 XOR	211
	F PRG TEST XOR	493
XPON	Returns exponent of number.  F MTH PARTS p.3 XPON	149
XRNG	Specifies x-axis display range PLOTR XRNG C → PLOT XRNG	295
<b>→</b> XRNG	Recalls x-axis display range to stack PLOTR  XRNG O PLOT  XRNG	293

Name, Key or Label	Description Type, Keys	Page
XROOT	Returns level 1 root of the real number in level 2.  A	134
XY	Selects x- and y-axis zoom.  O ZOOM XY	305
XYZ	Selects Rectangular mode.  MTH YECTR XYZ  O MODES p.3 XYZ	171
ΣX*Y	Returns sum of products of data in independent and dependent columns in \$\sum_DAT\$.  C \( \left\) STAT p.5 \( \Sigma \times \times \)	383
Y	Selects y-axis zoom. O ZOOM Y	305
ΣΥ	Returns sum of data in dependent column in \$\sum_DAT\$.  C \[ \] STAT p.5 \[ \SigmaY\]	383
ΣΥ^2	Returns sum of squares of data in dependent column in ΣDAT.  C STAT p.5 ΣΥ^2	383

Name, Key or Label	Description Type, Keys	Page
YCOL	Selects indicated column of \$\sum DAT\$ as dependent-variable column for two-variable statistics.  C  \[ \] STAT p.3 YCOL	376
→ YEOL	Recalls dependent-variable column number to stack.  O  TTAT p.3  YCOL	376
yd	International yard, length (.9144 m).  U ← UNITS LENG YD	
yd^2	Square yard, area (.83612736 m²).  U — UNITS AREA YD^2	
yd^3	Cubic yard, volume (.764554857984 m³).  U	
yr	Year, time (31556925.9747 s). U ← UNITS TIME YR	

Name, Key or Label	Description Type, Keys	Page
YRNG	Specifies y-axis display range.	293
	PLOTR YRNG	
	Recalls y-axis display range to stack.	293
	PLOTE PYRNG	200
	O PLOT PYRNG	
Z-B0X	Zooms in to box whose opposite corners	306
	are defined by mark and cursor.	
	AUTO Z-BOX	
	O GRAPH Z-BOX	
<b>(5) Z=30 X</b>	Zooms to box, autoscaling y-axis.	306
	DRAW (1Z-BOX	
	O GRAPH GZ-BOX	47

Name, Key or Label	Description Type, Keys	Page
ZOOM	Selects GRAPHICS ZOOM menu DRAW ZOOM AUTO ZOOM O SIGRAPH ZOOM	304
+	Adds two objects. A +	90
<b>*/</b> -	If cursor is on a number, changes sign of mantissa or exponent of that number. Otherwise, acts as NEG key. C +/-	47
+/-	Switches cursor style between superimposing and inverting cross.  DRAW p.3 +/-  RUTO p.3 +/-  O ← GRAPH p.3 +/-	302
+1-1	Add and subtract 1.  O — EQUATION — RULES +1-1	402
-	Subtracts two objects. A [-]	134

Name, Key or Label	Description Type, Keys	Page
()	Double negate and distribute.  O SEQUATION RULES -()	407
*	Multiplies two objects.  A 🗵	134
*1	Multiply by 1. O TEQUATION RULES *1	401
/	Divides two objects. A 🖹	134
/1	Divide by 1. O TO EQUATION TO RULES 71	401
^	Raises number to specified power.  A y <sup>z</sup>	134
^1	Raise to power 1.  O	401
<	"Less-than" comparison.  PRG TEST p.2  F \( \alpha \) 2	491

Name, Key or Label	Description Type, Keys	Page
≤	"Less-than-or-equal" comparison.  PRG TEST p.2   F   ☐ 4 3	491
>	"Greater-than" comparison.  PRG TEST p.2 > F \( \alpha \) \( \bar{P} \) \( \alpha \)	491
≥	"Greater-than-or-equal" comparison.  PRG TEST p.2   F	491
=	"Equals" function. A ← =	129
==	"Equality" comparison.  PRG TEST p.2 ==  F a 1	492

Name, Key or Label	Description Type, Keys	Page
<i>‡</i>	"Not-equal" comparison.  PRG TEST p.2 #  F @ 4 0	492
α	Turns alpha-entry mode on and off. O @	52
{}	Switches implicit parentheses on and off.  O	237
<b>*</b> **	Returns equation to stack as string.  O ( EQUATION )	230
0	Degree, plane angle (2.7777777778 × 10 <sup>-3</sup> ). U ← UNITS p.3 ANGL **	
	Factorial.  MTH PROB   F \( \alpha \) DEL	147

Name, Key or Label	Description Type, Keys	Page
30 p. S	Integral. A	428
д	Derivative. A → ②	419
Ω	Ohm, electric resistance (1 kg·m²/A²·s³).  U	
%	Returns level 2 percent of level 1.  A MTH PARTS p.2	138
π	Symbolic constant <i>π</i> (3.14159265359).	144
Σ	Summation. F → ∑	423
Σ+	Adds data point to matrix in ΣDAT.  C STAT Σ+	368
Σ-	Subtracts data point from matrix in $\Sigma DAT$ .  C $\blacksquare$ STAT $\blacksquare$ $\Sigma$ +	368
<b>√</b>	Returns square root of level-1 object.  A	134
	Appends local name, or variable of integration, and its value to evaluated expression.  F ALGEBRA p.2	416

Name, Key or Label	Description Type, Keys	Page
1/()	Double-invert and distribute.  O ( EQUATION   RULES   1/4 >	407
12/24	Switches between 12-hour and 24-hour display formats.  O	442
(())	Parenthesize neighbors.  O   RULES (())	403
</td <td>Expand-subexpression-left.  O</td> <td>404</td>	Expand-subexpression-left.  O	404
<b>→</b> (+	Executes (+ until no change in subexpression.	410
et de la ligera	O SEQUATION RULES	
	Distribute prefix function.  O ← EQUATION ← RULES → ()	405
+>	Expand-subexpression-right.  O	404
(* <b>•</b>	Executes →> until no change in subexpression.  O ← EQUATION ← RULES  → →>	410
<b>4-&gt;</b>	Commute arguments.  O ← EQUATION ← RULES ←→	404

Name, Key or Label	Description Type, Keys	Page
776	Creates local variables.  C → →	473
4	Left shift key.	52
<b>P</b>	Right shift key.	52
•	In command line, deletes character to left of cursor.  O •	47
	Deletes contents of current stack level.  O +STK •	72
	In multi-line command line: Moves cursor up one line.	75
	In Interactive Stack: Moves pointer up one level.	72
	In Graphics environment: Moves cursor up one pixel.	303
	In scrolling mode: Moves window up one pixel.	229
	In MatrixWriter application: Moves cell cursor up one row.	350
	In EquationWriter application: Starts numerator.	229
	In Selection environment: Moves cursor up one object. In catalogs: Moves pointer up one entry. O	399

Name, Key or Label	Description Type, Keys	Page
PA	In multi-line command line: Moves cursor to top line.	75
	In Interactive Stack: Moves pointer to highest numbered stack level.	72
	In Graphics environment: Moves cursor to top edge of <i>PICT</i> .	303
	In MatrixWriter application: Moves cell cursor to top element of current column.	350
, a''	In Selection environment: Moves cursor to topmost object.	399
	In catalogs: Moves pointer to top of list.  O	_
<b>4</b>	In catalogs: Moves pointer up one page. In Interactive Stack: Moves pointer up 4 levels.  O	72

Name, Key or Label	Description Type, Keys	Page
	In multi-line command line: Moves cursor down one line.	75
	In Interactive Stack: Moves pointer down one level.	72
	In Graphics environment: Moves cursor down one pixel.	303
	In scrolling mode: Moves window down one pixel.	229
	In MatrixWriter application: Moves cell cursor down one row.	350
	In EquationWriter application: Ends subexpression.	229
	In Selection environment: Moves cursor down one object.	399
	In catalogs: Moves pointer down one entry.  O	
	In multi-line command line: Moves cursor to bottom line.	75
	In Interactive Stack: Moves pointer to level 1.	72
	In Graphics environment: Moves cursor to bottom edge of <i>PICT</i> .	303
	In MatrixWriter application: Moves cell cursor to last element of current column.	350
	In EquationWriter application: Ends all subexpressions.	229
	In Selection environment: Moves cursor to bottommost object.	399
	In catalogs: Moves pointer to end of list.  O ♠▼	

Name, Key or Label	Description Type, Keys	Page
<b>€</b>	In catalogs: Moves pointer down page. In Interactive Stack: Moves pointer down 4 levels.  O    T  T  T  T  T  T  T  T  T  T  T  T	72
<b>4</b>	In command line: Moves cursor one character left.	75
1000	In Graphics environment: Moves cursor one pixel left.	303
	In scrolling mode: Moves window left one pixel.	229
	In MatrixWriter application: Moves cell cursor one column left.	350
esc we be	In EquationWriter application: Activates Selection environment.	398
(A)	In Selection environment: Moves cursor one object left.  O	399
<b>54</b>	In EquationWriter application and Graphics environments: Invokes scrolling mode.	229
( GRAPH)	O ( GRAPH)	303

Name, Key or Label	Description Type, Keys	Page
Pd	In command line: Moves cursor to start of current line.	75
	In Graphics environment: Moves cursor to left edge of <i>PICT</i> .	303
	In MatrixWriter application: Moves cell cursor to first element of current row.	350
	In Selection environment: Moves cursor to leftmost object.  O    O	399
	In command line: Moves cursor one character right.	75
	In Graphics environment: Moves cursor one pixel right.	303
	In scrolling mode: Moves window right one pixel.	229
	In MatrixWriter application: Moves cell cursor one column right.	350
	In EquationWriter application: Ends subexpression.	229
	In Selection environment: Moves cursor one object right.	399

Name, Key or Label	Description Type, Keys	Page
	In command line: Moves cursor to end of current line.	75
	In Graphics environment: Moves cursor to right edge of <i>PICT</i> .	303
	In MatrixWriter application: Moves cell cursor to last element of current row.	350
	In EquationWriter application: Ends all subexpressions.	229
	In Selection environment: Moves cursor to rightmost object.	399

# Index

At the shifted to be a second the case of	
aborting with the attention key	
command line, 54	
environments, 54	
programs, 54	
absolute value	
of a matrix, 360	
of a number, 148	
of complex numbers, 166	
of vectors, 177	
accented characters, generating,	
53	
accuracy	
in solving systems of equations,	al
362	al
of fraction conversion, 136	aı
of $\pi$ , 140	
adding	
a stack value to a variable, 115	
in the EquationWriter	
application, 231	
numbers, 134	
ADJST menu, 443	
Alarm Catalog, 449	
operations, 450	
alarms	
acknowledging, 446	
appointment, 445	
commands, 450	
control, 448	

execution action, 444 lost after recovering memory, 102 past due, 447 recovery from short-interval repeating alarms, 448 repeating, 444, 445 rescheduling, 447 reviewing and editing, 449 saving, 447 setting, 443 turning the beeper off, 447 unacknowledged, 447 used in programs, 453 ert annunciator, 48 gebra, 386-417 adding fractions, 409 building and moving parentheses, 403 collecting terms, 395, 402 commutation, association, and distribution, 404 comparing methods for isolating a variable, 393 expanding products and powers, expanding trigonometric functions, 409 general and principal solutions, 393

replacing in the EquationWriter isolating a variable, 389 limitations, 390 application, 248 short for algebraic objects, 85 moving terms, 402 simplification process, 128 multiple execution of Rules transformations, 410 stepwise evaluation, 126 using comparison functions in, rearrangement of exponentials, using complex numbers, 164 rearranging terms, 394 Rules transformations, 397 using complex numbers in, 161 Selection environment, 398 using logical functions in, 493 showing hidden variables, 394 using unit objects in, 191 viewing in the EquationWriter solving equations for a variable, application, 241 solving quadratic equations, 390 alpha key symbolic solutions, 388 activates alpha keyboard, 25 universal transformations, 400 press twice for alpha lock, 53 user-defined transformation, alpha keyboard, 52 alpha keyboard annunciator, 48 alpha left-shift keyboard, 50 ALGEBRA menu, 389, 395 Algebraic-entry mode, 76 alpha lock, 53, 222 annunciator, 48, 84 alpha right-shift keyboard, 50 entering unit objects, 189, 191 Alpha-entry mode, 52, 53, 222 ALRM menu, 444 Algebraic/Program-entry mode, ALRMDAT reserved variable, algebraics, 125-130 contains data for an alarm, are mathematical expressions, am/pm time format, 442 analytic functions, are a subset of collecting terms, 571 functions, 42 compared to programs, 125 differentiation, 419 disassembling, 90 with binary integers, 210 evaluation, 125 with tests, 493 evaluation of terms, 128 angle, in complex numbers, 157 mode for keying in, 76 angle conversion functions, 142 angle modes, 139, 170, 350 mode for keying into programs, 77 selecting, 139 angle units, 198 nested parentheses in, 128 object type number, 97 animation parentheses are highest of custom graphical image, 597 of Taylor's polynomials, 588 precedence in, 128 precedence of operators, 128 rearranging terms, 397

annunciators	ASCII Transmission mode, 617,
are displayed in status area, 48	629
complete list of, 48	assembling
share "territory" with messages,	complex numbers, 160, 166
48	unit objects, 206
answers to common questions, 656	vectors, 173, 183
antiderivative, 428	assigning user keys, 217
application cards, 651	association, algebra, 404
appointment alarms	attention key, 25
acknowledging, 446	halts current activity, 54
unacknowledged, 447	Automatic Alpha Lock mode, 222
approximation	automatic off, happens after 10
of symbolic constants, 144	minutes, 25
of the definite integral, 432	autoscaling a plot, 295
arc cosine, 140	available memory, number of
arc hyperbolic cosine, 137	bytes of unused user
arc hyperbolic sine, 137	memory, 101
arc hyperbolic tangent, 137	axes
arc sine, 140	labeling, 320
arc tangent, 140	specifying coordinates of
archiving memory, 624, 648	intersection, 320
area, beneath a plotted curve, 308	in a set a red to continue of
arguments, on the stack, 61	В
arithmetic	B
with a matrix and a vector, 356	backing up directories, 645
with complex arrays, 357	backspace editing
with complex numbers, 156	in EquationWriter application,
with dates, 454	241
with time, 456	in the command line, 75
with unit objects, 200	backup objects, 645, 646
with variables, 115	in custom menus, 213
with vectors, 353	object type number, 97
arithmetic and general math	store objects in plug-in memory,
functions, 134–135	89
arrays, 83, 346-364	bar over menu label, indicates a
assembling, 90	directory, 118
commands for, 360	bar plot, 379
complex, 357	from Plot application, 336
dimension (size), 90	from Statistics application, 379
entering using the command	base
line, 350	binary integers, 207
printing, 604	selecting, 208
	bace 10 antilogarithm 137

base 10 logarithm, 137 built-in commands, 90 base e (natural) antilogarithm, 137 object type number, 97 base e (natural) logarithm, 137 use 2.5 bytes, 101 base marker, 207 built-in constants, 144 built-in functions, 90 entering, 208 BASE menu, 82, 208, 210 compared to user-defined batteries, 25, 660 functions, 150 changing, 661 object type number, 97 for plug-in RAM, 638, 661 built-in menu, displaying, 534 for the HP 48, 661 built-in unit objects, 193 baud rate busy annunciator, 48 bytes command, returns during printing, 610 setting, 617 checksum, 101 beeper, turning off for alarm, 447 beeping, from a program, 522, 3 C Bessel functions, 585 best fit line, 376 cable connection, PC to HP 48, binary arithmetic, 207-211 binary base marker, 82 calculus, 418 - 436 binary integers, 82, 207 complete differentiation, 421 differentiation, 419 base, 207 differentiation of user-defined bits displayed, 208 functions, 422 calculations, 209 displaying, 208, 554 how the HP 48 does symbolic entering, 208 integration, 429 numerical integration, 432 internal representation, 208 logic commands for, 210 summations, 423 object type number, 97 symbolic integration, 428 Taylor's polynomial wordsize, 207 binary to real conversion, 210 approximation, 431 capital letters, 50 Binary Transmission mode, 617, carriage-return, dumping the print Black Gold Ltd, 27 buffer, 603 CASE...END program structure, blue keys, 25, 50 boolean logic commands, 210 Catalogs box, drawing, 337 Alarm, 449 brackets, used to enter vectors, Equation, 253, 258 172 BRCH menu, 494, 501 Review, 112 bubble sort, 561 Statistics, 370 centering a plot, 295 buffer length, serial I/O, 632 chain calculations, using the stack, buffered keystrokes, 48 buffered printing, 608

changing sign	collecting terms, 395
of a number, 47, 134	algebra, 402
changing the contents of a	column norm, of a matrix, 359
variable, 111	combinations, calculating, 147
character codes, 694-696	comma, as fraction mark, 58
character sets	command arguments on the stack,
printing the HP 48 character	61
set, 607	command line, 75-77
printing with the Infrared	cancelling with the attention
Printer, 609	key, 54
	editing in the EquationWriter
remapping the infrared printer, 603	application, 242
translating during input/output,	entering and editing text, 46, 75
626	entering arrays, 349
characters	keying in numbers, 47
converting numbers to	middle section of the display, 45
characters, 90	recovering previous command
determining their numeric	lines, 77
value, 90	scrolls after 21 characters, 46
entering special characters, 50	command-line string, building, 528
generating accents, 53	commands
checksum, 547	are a subset of operations, 42
used to verify objects, 101	as objects, 90
with input/output, 617	defined, 42
chi-squared test, 384	of one argument, 61
circle, drawing, 337	of two arguments, 62
clearing	common (base 10) antilogarithm,
alarms, 450	137
all variables in a directory, 115	common (base 10) logarithm, 137
flags, 222, 516	common variables, 105
last error, 542	commutation, algebra, 404
memory (press three keys), 101	compact format, of printed output
messages from the display, 48	604
objects when out of memory,	comparison functions, 491
103	in algebraics, 492
the stack, 64	complement, of a binary integer,
user key assignments, 219	210
using the attention key, 25	complex arrays
clock	arithmetic with, 357
adjusting, 443	commands for, 357
commands, 441	object type number, 97
recording execution time, 552	The gallular section (1916)
closing serial port, 614	
r,	

complex numbers, 81, 156 - 168 conversion, temperature, 197 allowed in algebraics, 161 converting binary to real, 210 arithmetic with, 156 arrays of, 357 complex array to real array, 357 complex to real, 90, 166 as the result of real-number operations, 163 compound unit to SI base units, assembling, 160, 166 196 changing angular modes, 157 date to number, 454 commands, 166 date to string, 454 compared to real numbers, 161 degrees to radians, 142 compared to vectors, 166, 167, HMS to number, 456 184 number to date, 454 conjugating, 166 number to HMS, 456 converting to real, 166 numbers to characters, 90 disassembling, 90, 160, 166 objects to a string, 554 display form, 158 objects to strings, 90 entering, 158 pixel coordinates to user-unit i (the imaginary number), 165 coordinates, 324 in expressions, 164 radians to degrees, 142 internal representation, 158 real array to complex array, 357 object type number, 97 real numbers to fractions, 136 printing, 604 real to binary, 210 complex to real, disassembling, 90 real to complex, 90, 166 conditional structures unit objects, 193, 194 in programs, 494, 499 units, 188, 195 CONIC plot type, 327 coordinate mode, changing, 171 conic plots, 329 coordinate pairs, can be conjugating represented by complex complex arrays, 357 numbers, 81 complex numbers, 166 coordinate systems for plots, 323 contents of a variable, 115 correcting typing mistakes, 47 connected plotting, 299 correlation, 377 constant matrix, calculating, 359 cosine, 140 constants, symbolic, 144 cotangent, creating a user-defined consumer price index, 364 function for, 151 continuing program execution, 483 counted strings, are counted after error, 541 sequences of characters, 86 covariance, 375, 376 continuous memory, not affected by ON / OFF, 25 cross product, 176, 353 contrast, adjusting, 25 CST menu, 213 control alarms, setting, 448 unit-object conversion in, 195 control codes, printing, 607 convergence, testing a series, 424

CST reserved variable contains data for custom menus,	days, between two dates, 455 debugging
108, 213	programs, 483
CTRL menu, 483	subroutines, 486
	decimal base marker, 82
current directory, 119	decimal numbers, 82, 207
current directory path, is displayed	decimal places, number displayed,
in status area, 48	58
current path, 119	decrementing
cursor keys, 27	
custom menus conversion of units, 195	the program loop counter, 513 time, 443
	defining
creating, 213	user-defined functions, 151
in programs, 535	variables, 107
menu labels, 213	•
shifted actions, 215	definite loops, 501
Customer Support, 656	degrees, converting to radians, 14.
customizing the calculator, 212-	Degrees mode, 139
223	delaying the print cycle, 603, 607
modifying the shift keys, 215	deleting
setting modes, 220	matrix row or column, 352
user key assignments, 216	tag from tagged object, 90
using system flags, 222	user key assignments, 219
Cylindrical mode, 170	delimiters
annunciator, 170	' delimits algebraic objects, 85
D	[ ] delimits arrays, 83, 173, 34
	( ) delimits complex numbers
darker contrast, 25	81, 158
data output, 531	
labeling with string commands,	« » delimits programs, 86, 468
532	" " delimits strings, 86
dates	: delimits tagged objects, 87
arithmetic with, 454	' ' prevents evaluation of a
changing format, 442	variable, 84, 112
commands, 441	# delimits binary integers, 82,
converting to numbers, 454	207, 208
converting to strings, 454	= delimits equations, 129
day/month/year date format,	_ delimits unit objects, 88, 187
442	delta days, number of days
month/day/year date format, 442	between dates, 454
setting, 441	
day/month/year date format, 442	

dependent variable	disassembling
not used for function plots, 299	complex numbers, 160, 166
plotting range for, 319	objects, 90
used for conic plots, 329, 333	unit objects, 206
used in statistics, 376	vectors, 173, 184
depth of stack, determining, 78	disconnected plotting, 299
derivatives	display
in the EquationWriter	adjusting contrast, 25
application, 233	clearing messages, 48
keying into the command line,	is divided into three sections, 45
420	status area, 48
plotting, 308	display modes
user-defined, 422	changing, 59
user-defined prefix is "der", 108	control format used to display
determinant, calculating, 359	numbers, 57
differentiation	displaying an object, from a
in one step, 421	program, 523
of algebraic expressions, 419	distribution, algebra, 404
of built-in functions, 150	dividing
of user-defined functions, 150	a variable by a stack value, 115
stepwise, 419	a vector into a matrix, 355
dimensionless units, 198	in the EquationWriter
directories	application, 231
concepts, 118	numbers, 134
contained in a variable, 110	do error, error trapping, 542
creating, 120, 123	dot product, 176, 353
current directory, 119	double-space printing, 606
determining all variables of a	DOUNTILEND program
specific type in, 98	structure, 510
directory path, 119	dropping
HOME directory, 118	the stack, 64, 71, 78
new variables are added to the	duplicate variable names, allowed
current directory, 121	in different directories, 121
object type number, 97	duplicating
parent directory, 119	level 1 in the stack, 65, 71
purging, 123	objects on the stack, 78
recalling, 123	objects on the stack, 70
searching directories for a	- transfer at the street formation has
variable name during	E
evaluation, 121	e, is a built-in constant, 144
switching up a level, 122	echoing stack contents, 71
dimental 110	EDIT

EDIT menu, 68

directory path, 119

is displayed in status area, 48

editing solving for a variable, 386 equations in the EquationWriter solving quadratics, 389 solving with the Plot application, application, 240 in the command line, 75 elapsed time, calculating, 457 used to create a user-defined Engineering mode, 58 function, 151 EquationWriter application, 24, enter key, 25 227 - 250duplicates level 1, 65 entry modes addition, subtraction, and multiplication, 230 for entering matrices, 351 backspace editing, 241 four types, 76 environmental limits, plug-in building unit objects, 204 command line editing, 242 cards, 660 environments creating equations, 230 Alarm Catalog, 449 derivatives, 233 are cancelled with the attention division and fractions, 231 key, 54 editing equations, 240 editing subexpressions, 243 Equation Catalog, 258 Graphics, 286, 300 exponents, 232 how it is organized, 228 Interactive Stack, 70 implicit parentheses, 229 Selection, 244, 398 Statistics Catalog, 371 inserting objects from the stack, EQ reserved variable 246 contains the current equation, integrals, 234 108, 253, 286 keyboard operation, 229 equal to, comparison test, 491 numbers and names, 230 Equation Catalog, 253, 258 powers of 10, 233 commands, 259 replacing subexpressions, 247 Selection environment, 243 creating a list of equations, 274 exiting, 262 square root and x-th root, 232 linking equations, 272 summations, 235 reordering, 259 unit objects, 235 using parentheses, 233, 236 equation to stack, disassembling, viewing algebraics and unit equations objects, 240 can be arguments to a function, where function, 236 erasing PICT, 292, 323 contain an "=" sign, 129 error messages, are displayed in editing in the EquationWriter status area, 48 application, 242 error recovery, from accidentally general and principal solutions, purging a variable, 115 393 linking, 272

expanding products and powers, errors clearing last, 542 continuing program execution exponent after, 541 display format, 58 error message, 542 extracting from a number, 148 error number, 542 in the EquationWriter error trapping commands, 542 application, 232 returning most recent Kermit keying in, 47 error, 614 exponential functions, 137 trapping, 541 exponentials, rearrangement using user-defined, 546 algebra, 408 escape sequences, printing, 607 expressions etcetera kev do not contain an "=", 129 used to enter accented using complex numbers in, 164 characters, 53 used to enter special characters, F Euclidean norm, calculating, 359 F test, 385 evaluation factorial, 147 is affected by results mode, 127 factoring unit expressions, 199 false, result of a test, 490 of a variable, 109 of a variable containing a FCN menu, 308 program, 110 Fibonacci numbers, 548 of algebraics, 125, 126 file names, PC versus HP 48, 628 of local variables, 476, 569 files, sending and receiving, 614 of string contents, 90 finishing server mode, 614 of symbolic constants, 145 finite series, 423 of variables prevented by first order equation, solving for x, quoting, 84, 112 392 the precedence of operators Fix mode, 58 determines the order of flags, 515 evaluation of terms, 128 complete list of, 699 evaluation of variables, searching I/O Device, 610 directories for the variable Line-feed, 608 name, 121 Printing Device, 610 exclusive or recalling and storing, 518, 556 with binary integers, 210 setting, clearing, and testing, with tests, 493 222, 516 executing that control the evaluation of commands and functions from symbolic constants, 145 the stack, 61 formal variable, does not contain programs, 472 an object, 152 user-defined functions, 152

format of numbers in the display, 57 of printed output, 604	using symbolic arguments, 149 future date, calculating, 455
FORNEXT loop, 506	G
FORSTEP loop, 508	G
fraction approximation, of a number, 134	general solutions, of an equation, 393
fraction conversion	geometric series, 424
accuracy of result, 136	getting files, input/output, 614
functions, 136	getting the <i>n-th</i> array element, 90
fraction mark, 58	global names, object type number,
fractional part, math function, 148	97
fractions	global variables, 105
adding using algebra, 409	Grads mode, 139
in the EquationWriter	annunciator, 48
application, 231	Graphics environment, 300
free memory, number of bytes of	adding graphical elements to
unused user memory, 101	PICT in, 337
freeing memory, 649	analyzing p! tted functions in,
freezing part of the display, 523	306
frequencies, in statistical samples,	introduced, 286
374	stack-related operations, 3412
Frobenius norm, calculating, 359	zoom operations in, 304
function arguments on the stack,	GRAPHICS FCN menu, 308
61	graphics objects
FUNCTION plot type, 327	in programs, 342
function plots, 328	introduced, 287
functions	manipulating on the stack, 342
analyzing in the Graphics	object type number, 97
environment, 306	printing, 606, 610
angle conversion, 142	size, 90
are a subset of commands, 42	stack form, 340
as objects, 90	store pictures, 87
built-in, 150	GRAPHICS ZOOM menu, 305
creating user-defined functions, 151	greater than, comparison function 491
defined, 42	greater than or equal to,
math, 132-149	comparison function, 491
on the keyboard, 134-135	greatest integer, math function,
plotting, 328	148
user-defined, 150	Greek letters, entering from the
using equations as arguments,	keyboard, 50

## H

halt annunciator, 48 halting programs with the attention key, programs with the HALT command, 483, 523 the root-finder, 277 hexadecimal base marker, 82 hexadecimal numbers, 82, 207 hidden variables, showing, 394 histogram plot from Plot application, 336 from Statistics application, 378, 382 HMS format, 456 HOME, is power-on directory, 48 HOME directory, 118, 124 selecting, 122 HP Solve application, 24, 250-282 choosing guesses, 266 consists of two menus, 253 customizing the SOLVR menu, 269 editing equations, 256 entering a new equation, 257 finding solutions of programs, 275 how it works, 276 interpreting results, 279 multiple solutions, 266 no solution found, 282 plotting solutions, 266 recalling equations, 256 sign reversal, 280 solving equations, 254, 256 solving expressions, 254 solving programs, 254 specifying an equation from the Equation Catalog, 258 specifying the current equation. 255

storing equations, 256
used with Plot application, 252
using unit objects with, 267
verifying solutions, 265
humidity, effect on calculator, 660
HYP menu, 137
hyperbolic cosine, 137
hyperbolic functions, 137
hyperbolic sine, 137
hyperbolic tangent, 137

i (the imaginary number), 144, 165 ideal gas equation, 185 identity matrix, calculating, 359 IFERR...THEN...ELSE...END error trap for programs, 544 IFERR...THEN...END error trap for programs, 542 IFT if-then-end function, 499 IFTE if-then-else function, 500 IF...THEN...ELSE...END structure for programs, 496 IF...THEN...END structure for programs, 494 imaginary part, 166 of a complex array, 357 immediate execution of variables, Immediate-entry mode, 76 entering unit objects, 188 incrementing the program loop counter, 513 time, 443 indefinite loops, 510 independent memory, 642 independent variable plotting range for, 319 specifying for plots, 294 statistics, 376

Infrared Printer, 602, 609 building the command-line character sets, 607, 609 string, 528 displaying a built-in menu, 534 testing, 670 Infrared Transmission mode, 617 displaying objects, 523 freezing part of the display, 523 input halting programs, 523 options, 526 prompting for data input, 524 labeling program output, 531 input/output, 612-634 options for the input command, Binary/ASCII modes, 629 526 prompting for data input, 524 cable connection, 621 commands for, 614 prompting for input, 520 returning a key location, 539 downloading data, 612 HP 48 to HP 48, 613, 619 using custom menus, 535 using string commands to label Kermit file transfer protocol, data output, 532 local/local configuration, 620 using tagged objects as data local/server configuration, 620 output, 531 PC to HP 48, 621, 623 Interactive Stack, 70 – 75 serial commands for, 632 activating, 70 serial loop back test, 671 exiting, 74 setting I/O parameters, 617 operations, 71 translating character codes, 626 viewing objects in, 73 types of data allowed, 613 internal representation inserting, matrix row or column, binary integers, 208 vectors, 171 insufficient memory, error International System of Units (SI), message, 103 integer part, math function, 148 inverse integrals of a matrix, 354 in the EquationWriter of a number, 134 application, 234 of a variable, 115 keying into command line, 428 inverse hyperbolic cosine, 137 integrand, approximation, 431 inverse hyperbolic sine, 137 integration inverse hyperbolic tangent, 137 accuracy factor, 433 I/O Device flag, 610 I/O menu, 614, 632 from the stack, 436 I/O SETUP menu, 617 how the HP 48 does it, 429 numerical, 432 IOPAR reserved variable symbolic, 428 stores I/O parameters, 108, 618 interactive programs, 519-540 isolating a variable, algebra, 389 iterative refinement, solving beeping, 522 building a temporary menu, 539 systems of equations, 362

Joe's grocery, 596

#### K

keeping the stack, 71 Kermit file transfer protocol, 612 Kermit modes, server/local, 616 Kermit protocol commands, 614 key assignments, user keyboard, 217 key location, returning, 539 keyboard blue keys, 50 clearing key assignments, 219 entering letters, 52 entering special characters, 54 Greek letters, 50 has six levels, 25, 50 keying in a program, 470 keying in accented characters, keying in dates, 441 keying in delimiters, 55 keying in numbers, 47 keying in statistics data, 369 keying in time, 442 keying in vectors, 172 lowercase letters, 50, 52 number pad, 50 orange keys, 50 queues 15 keystrokes, 48 redefining, 216 shift keys, 52 special characters, 50 uppercase letters, 50, 52 using backspace to erase mistakes, 47 keyboard functions, 134-135 keyboard layout, 26

#### Ļ

labeling data output with string commands, 532 plot axes, 320 program output, 531 largest real number, 81 last argument restores arguments after insufficient memory condition, 103 used to recover purged variable, 115 last argument key, 64 last command key, 77 last menu key, 57 left-shift annunciator, 48, 52 left-shift key, 52 activates left-shift keyboard, 25 length, of a vector, 353 less than, comparison function, less than or equal to, comparison function, 491 letters entering, 52 generating accents, 53 lowercase, 52 uppercase, 52 levels of the stack, 46 returning current level number, library commands, 653 LIBRARY menu, 651 library objects, 651 attaching to a directory, 651 contain commands and operations, 89 object type number, 97 lighter contrast, 25 line, drawing, 337 line length, during printing, 610

keystroke queue, 48

line termination, during printing, FOR...STEP, 508 increment loop counter, 513 610 START...NEXT, 501 linear equations, 357 START...STEP, 504 accuracy of solution, 362 linear regression, 377 WHILE...REPEAT...END. line-feed, dumping the print 511 buffer, 603 low battery (alert) annunciator, 48 low memory, 102, 103 Line-feed flag, 608 low-battery condition, replacing linking equations in the Equation batteries, 660 Catalog, 272 listing the stack, creates a list of lowercase alpha lock, 53 lowercase letters, 50, 52 objects, 71 lists are sequences of objects, 86 М assembling, 90 magnitude, of complex numbers, creating a subset, 90 157 mode for keying in, 77 mantissa number of elements (size), 90 display format, 58 object type number, 97 extracting from a number, 148 position of an object in, 90 keying in, 47 put replaces *n-th* element, 90 mark, defines a position in PICT, replace a sub-list, 90 Local mode, 616 math functions, 132 – 149 local names, object type number, with vectors, 177 MATR menu, 359 local variables, 105 matrices, 83 evaluation, 476 adding and subtracting, 354 scope of definition, 476 are arrays, 345 used in programs, 473 arithmetic with vectors, 355 local/local configuration commands for, 359 HP 48 to HP 48, 620 complex, 357 PC to HP 48, 623 determinant of, 359 local/server configuration dividing by a vector, 355 HP 48 to HP 48, 620 editing, 350 PC to HP 48, 624 logarithmic functions, 137 identity, 359 keying in, 346 logic commands, 210 norms of, 359 logical functions, 493 product of, 354 in algebraics, 493 put replaces n-th element, 90 loops, 501 decrement loop counter, 513 reciprocal, 354 redimensioning, 359 DO...UNTIL...END, 510 FOR...NEXT, 506

254	CD ADVINCE FOUL AND	
scalar multiplication, 354	GRAPHICS FCN, 308	
transposing, 359	GRAPHICS ZOOM, 305	
MATRIX menu, 346	I/O, 614, 632	
MatrixWriter application, 346	I/O SETUP, 617	
deleting row or column, 351	LIBRARY, 651	
entering arrays, 350	MATRIX, 347	
entering statistical data, 370	MEMORY, 101	
entry modes for entering	MEMORY Arithmetic, 115	
matrices, 351	MODES, 57	
inserting row or column, 351	MODES Customization, 220	
maximum, math function, 148	MTH, 133	
maximum value, of a sample, 374	MTH BASE, 82, 208, 210	
MAXR, is a built-in constant, 144	MTH HYP, 137	
mean, of a sample, 374	MTH MATR, 359	
median	MTH PARTS, 138	
of a list, 563	MTH PROB, 147, 383	
of statistics data, 560	MTH VECTR, 142, 172, 183	
memory	PLOT, 290	
amount used by objects, 101	PLOTR, 292	
archiving, 624, 648	PRG BRCH, 494, 501	
backing up, 624	PRG CTRL, 483	
cancelling clearing operation,	PRG OBJ, 90	
102	PRG STK, 78	
checksum of an object, 101	PRG TEST, 491	
clearing, 101, 102	PRINT, 603	
expanding, 100	SOLVE, 253, 256	
freeing merged memory, 649	<b>SOLVE SOLVR, 253, 263</b>	
insufficient memory, 103	STAT, 367	
low-memory conditions, 102	STAT MODL, 376	
no room for last stack, 102	TIME, 440	
no room to show stack, 103	TIME ADJST, 443	
not affected by ON / OFF, 25	TIME ALRM, 444	
number of bytes unused, 101	TIME RPT, 445	
out of memory, 103	TIME SET, 441	
RAM and ROM, 100, 635	UNITS Catalog, 187, 188, 193	
restoring backed up user	UNITS Command, 187	
memory, 625	VAR, 106, 112, 118	
MEMORY Arithmetic menu, 115	menu keys, 55	
MEMORY menu, 101	menu labels	
menu descriptions	bar indicates a directory, 118	
ALGEBRA, 389, 395	describe menu keys, 45, 55	
CST, 213	in custom menus, 213	
EDIT, 68	variable names, 106, 108	
	, minore manies, 100, 100	

Standard, 58 menus bar indicates sub-menu, 56 Symbolic Evaluation, 223 cycling multiple pages, 56 Symbolic Results, 127, 144 User, 216, 223 define menu keys, 55 leaving, 56 Wire Transmission, 617 selecting, 56 model, in Statistics application, selecting next and previous, 56 switching to last menu, 57 modes used in programs, 534 changing, 554 changing coordinate mode, 171 merged memory, 642 messages, 677-693 for printing, 607 are displayed in status area, 48 reset by clearing memory, 101 clearing from the display, 48 selecting, 220 share "territory" with setting, 57 using system flags to set, 222 annunciators, 48 minimum, math function, 148 MODES Customization menu, minimum value, of a sample, 374 220 MODES menu, 57 MINR, is a built-in constant, 144 mod (modulo), math function, 148 modes of entry, four types, 76 mode, changing, 77 MODL menu, 376 mode types month/day/year date format, 442 Algebraic-entry, 76, 190 most significant bits, binary Algebraic/Program-entry, 77 integers, 208 Alpha-entry, 52, 53, 222 moving, the stack pointer, 71 ASCII Transmission, 617, 629 moving terms, algebra, 402 Automatic Alpha Lock, 222 MTH BASE menu, 82, 208, 210 MTH HYP menu, 137 Binary Transmission, 617, 629 Cylindrical, 170 MTH MATR menu, 359 Degrees, 139 MTH menu, 133 Engineering, 58 MTH PARTS menu, 138 Fix, 58 MTH PROB menu, 147, 384 Grads, 139 MTH VECTR menu, 142, 171, Immediate-entry, 76 183 Infrared Transmission, 617 multiline format, of printed Local, 616 output, 604 Numerical Results, 127, 144 multiplying Polar, 81, 157, 170 a variable by a stack value, 115 Program-entry, 77, 470 in the EquationWriter Radians, 139 application, 230 Rectangular, 81, 157, 170 numbers, 134 Scientific, 58 Server, 614, 616

Spherical, 170

ı	k	ı	ı
	١	٩	ı
-		٩	

names

are used to identify variables, 84 contained in a variable, 110 in the EquationWriter application, 230 reviewing unit names, 191 naming variables, 108 natural (base e) antilogarithm, 137 natural (base e) logarithm, 137 negating complex numbers, 166

contents of a variable, 115 negative numbers, keying in, 47 nested loops, 561

nested parentheses, in algebraics, 128

nesting, user-defined functions, 153

next key, selects next menu, 56 no room for last stack, error message, 102

no room to show stack, error message, 103

normal distribution, 385 not

with binary integers, 210 with tests, 493 number pad, of the keyboard, 50 numbers

converting to a character, 90 converting to date, 454 display modes, 57 in the EquationWriter application, 230 internal representation, 57 keying into the command line,

numerator, in the EquationWriter application, 229 numerical constants, 144 numerical integration, 432 accuracy factor, 433 Numerical Results mode, 127, 144 numerical value of a character, 90

#### 0

OBJ menu, 90 object to string, converting, 90 object type number, 97 determining, 97 object types, 80, 97 arrays, 83 backup objects, 89 binary integers, 82 built-in commands, 90 built-in functions, 90 complex numbers, 81 counted strings, 86 directories, 89 graphics objects, 87 library objects, 89 lists, 86 matrices, 83 names, 84 programs, 85 real numbers, 81 strings, 86 tagged objects, 87 unit objects, 88 vectors, 83 XLIB names, 89 objects are delimited by punctuation characters, 55 checksum, 101 disassembling, 90 inserting from the stack into the EquationWriter application,

manipulation commands for, 90

viewing in the Interactive Stack,

viewing and editing, 66, 67

73

octal base marker, 82 PARTS menu, 138 octal numbers, 82, 207 past due alarms, 447 off key, 25 path, returning current directory path, 120 on key, 25 becomes the attention key, 54 PC file names versus HP 48 file one-argument commands, 61 names, 628 one-dimensional vectors, 83 PC to HP 48 opening serial port, 614 cable connection, 621 operations, defined, 42 Input/Output, 621 percent calculations, with unit objects, 202 with binary integers, 210 percent change, calculating, 138 with tests, 493 percent of total, calculating, 138 orange keys, 25, 50 out of memory, 103 period, as fraction mark, 58 permutations, calculating, 147 output, 531 over-determined systems, 363 photometric units, 198 overflow, real numbers, 81 pi, 140 picking an object from stack, 78 picking stack contents, 71 **PICT**  $\pi$ , is a built-in constant, 144 adding graphical elements to, pacing (receive/transmit), setting, 336 changing the size of, 325 packet, sending commands to a erasing, 292 server, 614 erasing and restoring to its packets, sending commands to a default size, 323 server, 631 stack manipulation of, 341 paired-sample statistics, 375 pixel coordinates in plots, 323 PARAMETRIC plot type, 327 plane angles, 198 parametric plots, 331 Plot application, 24, 283-344 parent directory, 119 contains two menus and special parentheses environment, 286 are highest precedence in data elements in, 286 algebraics, 128 structure of, 286 delimit complex numbers, 81 used with HP Solve application, used in algebra, 403 252 used to enter complex numbers, PLOT menu, 290 158 PLOT PLOTR menu, 292 using in the EquationWriter plot types, 327 application, 229, 233, 236 BAR, 329, 336 parity CONIC, 327, 328 during printing, 610 **FUNCTION, 328, 329** setting, 617, 619 HISTOGRAM, 328, 336

PARAMETRIC, 327, 333 two or more equations, 300 POLAR, 327, 330 unit objects in, 335 SCATTER, 328, 336 user-unit and pixel coordinates, TRUTH, 327, 333 323 plotting what the HP 48 can plot, 283 analyzing plotted functions, 307 with autoscaling, 295 axes labels and intersection, 320 with specified y-axis range, 295 conic plots, 329 working with difficult plots, 314 connected and disconnected x-axis display range, 295 plotting, 300 y-axis display range, 295 coordinate systems for, 324 zoom operations, 304 function plots, 328 zoom-to-box, 306 how DRAW plots points, 298 zoom-to-box with autoscaling, paired-sample statistics, 375 306 parameters stored in PPAR, 322 plotting range parametric plots, 332 specifying, 319 plotting range of independent valuable for parametric and and dependent variables, truth plots, 321 320 plug-in cards, 635 polar plots, 331 environmental limits, 660 programs and user-defined installing and removing, 636 functions, 335 plug-in RAM, 100 refinement options for, 318 plug-in RAM batteries, 661 resetting plot parameters, 292 plug-in ROM, 100 resetting plot parameters and polar angle, 166 erasing PICT, 323 Polar mode, 81, 157, 170 resolution, 321 annunciator, 157, 170 single-sample statistics, 374 POLAR plot type, 328 size of *PICT*, changing, 325 polar plots, 331 specifying independent variable, Polar/Cylindrical Coordinates mode, annunciator, 48 specifying plot parameters, 291 Polar/Spherical Coordinates specifying the center and scale, mode, annunciator, 48 population statistics, 375 statistical data from the Plot port RAM test, 669 application, 335 position of object in list, 90 statistics, 379 power conservation, automatic off status message indicates plot after 10 minutes, 25 parameters, 291 power-on directory, is HOME, 48 powers of 10, in the the derivative of a plotted function, 308 EquationWriter application, truth plots, 333 233

PPAR reserved variable printing, 602-611 contains Plot parameters, 108, PROB menu, 147, 383 probability, 147 precedence of functions producer price index, 364 in algebraics, 128 product of matrices, 354 in unit objects, 191 products and powers, expanding, precision, of displayed number, 58 predicted value, 376 program execution, continuing prefixing user-defined units, 206 after error, 541 previous key program-entry annunciator, 470 right-shift goes to first page, 56 Program-entry mode, 77, 470 selects previous menu, 56 annunciator, 48 previous results, used in chain entering unit objects, 188 calculations, 62 programming examples, 547 - 599 PRG BRCH menu, 494, 501 programs PRG CTRL menu, 483 aborting with the attention key, PRG OBJ menu, 90 PRG STK menu, 78 are sequences of commands, 85, PRG TEST menu, 491 primary (unshifted) keyboard, 25, as arguments, 569 calculating execution time, 551 principal solutions, of an equation, CASE...END structure, 497 compared to algebraics, 125 PRINT menu, 603 conditional structures, 494 printing continuing execution, 483 accumulating data in the buffer, data input commands for, 520 608 DO...UNTIL...END structure, and the HP 48 character set, 607 510 double spacing, 606 editing, 472 escape sequences and control evaluating variables containing sequences, 607 programs, 110 graphics objects, 606, 610 evaluation of local names, 476 modes, 607 executing, 472 PRTPAR contains printer finding solution with the HP parameters, 610 Solve application, 275 setting the delay, 607 FOR...NEXT structure, 506 strings, 606 FOR...STEP structure, 508 testing, 670 halting, 483 the display, 605 IF...THEN...ELSE...END the stack, 606 structure, 496 to the serial port, 609 IF...THEN...END structure, variables, 606 494 Printing Device flag, 610 input/output, 519

keying in, 470 loop structures, 501 quadratic equations mode for keying in, 77 solving, 389, 390 object type number, 97 queued keystrokes, 48 plotting, 334 quotes, used to prevent evaluation scope of local variables, 476 of a variable, 112 single-step execution, 483 START...NEXT structure, 501 R START...STEP structure, 504 radians, converting to degrees, 142 suspending execution with the Radians mode, 139 WAIT command, 534 annunciator, 48 that act like user-defined radix mark. See fraction mark functions, 478 **RAM** that manipulate data on the also known as user memory, 100 stack, 479 can be expanded with plug-in used by other programs, 582 cards, 100 using alarms in, 453 memory which can be altered, using custom menus in, 535 100 using local variables in, 473 (random-access memory), 635 using subroutines in, 480 RAM cards, 638 using tests in, 490 batteries, 638 WHILE...REPEAT...END expanding user memory, 643 structure, 511 installing and removing, 636 working with graphics objects, used for backup, 644 342 write-protect switch, 641 prompting for input, 520 random number, selecting, 147 PRTPAR reserved variable range of values, real numbers, 81 contains printer parameters, 610 real arrays, object type number, 97 contains printing parameters, real numbers, 81 108 compared to complex, 161 pseudo-random number, 147 converting to complex, 166 punctuation characters, as converting to fractions, 136 delimiters, 55 display format, 58 purging MAXR and MINR, 144 alarms, 450 object type number, 97 backup objects, 646 overflow, 81 directories, 123 range of values, 81 objects when out of memory, underflow, 81 real part variables, 114 of a complex array, 357 put element into array, 90 of a complex number, 166 real to complex, assembling, 90

restoring backed up user memory, rearranging terms, the Rules transformations, 397 results, on the stack, 61 recalling Review Catalog, 112 contents of a variable, 110 right-shift annunciator, 48, 52 flags, 518 right-shift key, 52 user key assignments, 220 activates right-shift keyboard, 25 receive pacing, setting, 619 receiving data, serial I/O, 614 right-shift keyboard, 50 receiving strings, serial I/O, 632 rolling the stack, 71, 78 reciprocal, of a unit object, 201 ROM can be expanded with plug-in recover memory, cancelling cards, 100 clearing operation, 102 memory which cannot be recovering last arguments, 64 altered, 100 previous command lines, 77 (read-only memory), 635 ROM cards, installing and Rectangular mode, 81, 157, 170 removing, 636 annunciator, 158 recursion, calculating Fibonacci root numbers, 548 finding the square or x-th root of a number, 134 redefining the keyboard, 216 of a plotted function, 308 redimensioning a matrix, 359 registers, variables used instead of, root-finder 105 halting, 277 in the HP Solve application, 276 regulatory information, 676 intermediate guesses, 278 reordering using initial guesses, 277 Equation Catalog, 259 rotate commands, with binary Statistics Catalog, 371 integers, 210 the VAR menu, 113 repair, 674 rotating the stack, 78 replace part of a list or string, 90 rounding errors, solving systems of replacing batteries, 660 equations, 361 rescheduling alarms, 447 rounding numbers, 148 reserved variables, 108 row norm, calculating, 359 RPT menu, 445 resetting rules of precedence, in algebraics, memory, 101 plot parameters, 292, 323 resolution Rules transformations, 397–417 how it affects statistical plots, examples, 400 executing a transformation, 399 321 exiting a RULES menu, 400 specifying for plots, 320 speeding up plots by increasing, selecting, 399

845

S
sample statistics, 374
scalar multiplication, matrices, 354
scaling a plot, 295
scatter plot
from Plot application, 336
from Statistics application, 378
Scientific mode, 58
scientific numbers, keying in
exponent and mantissa, 47
scope of local variables, 105, 476
scrolling
of the command line, 46
the stack, 66
$\Sigma DAT$ reserved variable
contains current statistical
matrix, 108, 369
seed for random number, 147
Selection environment, 243, 398
editing subexpressions, 244
self-test, 667
sending a serial break, 632
sending data, serial I/O, 614

98 serial cable, PC to HP 48, 621 serial I/O commands, 632 serial loop-back test, 671 serial port

separating variable names by type,

configuring for printing, 610 opening and closing, 614 printing, 609

Server mode, 614, 616 starting and finishing, 614 Service, 674

testing calculator operation, 665 SET menu, 441

setting

display I/O parameters, 614 flags, 222

serial I/O timeout, 632 SETUP menu, 617 shift commands, with binary integers, 210 shift keys, 25, 52 in custom menus, 215

press twice to cancel, 52 short-interval repeating alarms,

showing hidden variables, 394 sign

changing the sign of a number,

determining, 148 of a unit object, 203 significant digits, 58 simplification of algebraics, 128

sine, 140 single-sample statistics, 374 single-step

execution of a program, 483 program operations, 483 size

of a graphics object, 90 of a list or string, 90 of an array (dimension), 90 of *PICT*, 325

slope, of a plotted function, 308 smallest integer, math function,

smallest real number, 81 Snedecor's F test, 384 solid angles, 198 SOLVE menu, 253, 256 SOLVE SOLVR menu, 263 Solver-list, naming, 270 solving

for a variable, 388 quadratic equations, 389, 390 systems of equations, 356

SOLVR menu, 253, 263 customizing, 269

ΣPAR reserved variable contains Statistical parameters, 108, 378

special characters	stack to unit, assembling, 90
entering from the keyboard, 50	standard deviation, 374, 375
table of, 54	Standard mode, 58
Spherical mode, 170	STARTNEXT definite loops,
square matrix, inverting, 354	501
square root	STARTSTEP definite loops,
in the EquationWriter	504
application, 232	STAT menu, 367
of a number, 134	STAT MODL menu, 376
squaring a number, 134	statistics
stack	dependent variable, 376
clearing, 64	designating the current matrix,
commands, 78	369
dropping, 64	editing data, 370
duplicating level 1, 65	entering data, 368, 369
Graphics environment	independent variable, 376
operations, 341	manipulating data, 368
inserting level 1 into the	paired-sample statistics, 375
EquationWriter application,	plotting samples, 378
246	population statistics, 375
is a sequence of storage	sample statistics, 374
locations, 46, 60	summation commands, 383
levels, 46	Statistics Catalog, 370
lost after recovering memory,	operations, 371
102	reordering, 371
no room to show, 103	statistics, 364-385
one-argument commands, 61	status area
ordinary calculations, 61	displays current path, 119
printing, 606	of the display, 48
recovering last arguments, 64	stepwise differentiation, 419
splitting equations, 90	STK menu, 78
stores graphics objects, 87	storage locations
swapping levels 1 and 2, 63	the stack, 46, 60
two-argument commands, 62	storing
using previous results, 62	flags, 518
viewing and editing objects, 67	user keys, 217
viewing and editing variables, 67	variables, 107
stack display, is divided into three	strings
sections, 45	are sequences of characters, 86
stack pointer, moving, 71	combining, 90
stack to array, assembling, 90	counted strings, 86
stack to list, assembling, 90	executing contents of, 90
stack to tag, assembling, 90	from an object, 90
=	- · · · · · · · · · · · · · · · · · · ·

making a subset, 90 symbolic arguments, used in number of characters (size), 90 functions, 149 object type number, 97 symbolic constants, 144 position within another string, converting to values, 144 e, 144 printing, 604, 606 evaluation, 145 replacing a sub-list, 90 i (the imaginary number), 144, Student's t test, 384 165 subdirectories, 118  $\pi$ , 140, 144 can be manipulated like other Symbolic Evaluation mode, 223 variables, 124 symbolic integration, 428 creating, 120 symbolic math, 24 evaluating its name to switch to Symbolic Results mode, 127, 144 it, 122 syntax subexpressions of an integral, 428 completed with cursor keys in of variable names, 108 the EquationWriter unit objects, 187 application, 229 user-defined function, 154 defined, 243, 395 system flags, 222, 515 editing in the EquationWriter complete list of, 699 application, 243 systems of equations, 356 replacing in the EquationWriter accuracy of solution, 361 application, 247 over-determined, 362 the Selection environment, 398 under-determined, 362 subroutines, 480 single-step execution, 486 subset of a list or string, 90 t test, 384 subtracting a stack value from variable, 115 tagged objects in the EquationWriter are labeled objects, 87 application, 230 as data output, 531 assembling from the stack, 90 numbers, 134 summation statistics, 383 deleting the tag, 90 disassembling, 90 summations, 423 calculated from the stack, 426 object type number, 97 entering, 423 useful for labeling, 88 in the EquationWriter tangent, 140 Taylor's polynomials application, 235 suspending a program, 534 approximation of the integrand, swapping levels in the stack, 63 switching to the parent or HOME computing for an algebraic, 426 directory, 122 translating point of evaluation, 427

temperature, effect on calculator,	total, of a sample, 374 translating characters,
temperature conversion, 197	input/output, 626
temporary menu, used in	translating input/output, 617
interactive programs, 539	translation mode, during printing,
temporary variables, 105 used in programs, 473	610 transmit pacing
TEST menu, 491	during printing, 610
test statistics, 383	setting, 619
testing	transmitting, serial I/O, 632
calculator operation, 665	transmitting annunciator, 48
flags, 222, 516	transpose, calculating, 359
Infrared Printer, 670	trigonometric functions, 140
keyboard operation, 667	expanding using algebra, 409
port RAM test, 669	trigonometric operations, with unit
self-test, 667	objects, 203
serial loop back test, 671	true, result of a test, 490
text, entering and editing in the	truncating numbers, 148
command line, 46	TRUTH plot type, 327
ticks, 457	truth plots, 333
system time as a binary integer,	twelve-hour time format, 442
456	twenty-four hour time format, 442
time	two-dimensional points, can be
adjusting, 443	represented complex
am/pm time format, 442	numbers, 81
changing format, 442	two-dimensional vectors, 83
commands, 441	type
required to execute a program, 551	returning object type number, 97, 493
setting, 442	
twelve-hour time format, 442	U
twenty-four hour time format,	the define plant as compl.
442	under-determined systems, 363
TIME ADJST menu, 443	underflow, real numbers, 81
TIME ALRM menu, 444	unemployment rate, 364
time arithmetic, 456	unit objects
TIME menu, 440	are numbers combined with
TIME RPT menu, 445	unit, 88
TIME SET menu, 441	assembling from the stack, 90
timeout	disassembling, 90
automatic off after 10 minutes,	in custom menus, 213
25	in HP Solve application, 267
setting, 632	

in the EquationWriter unit object conversion, 194 application, 235 unit object conversion in the object type number, 97 CST menu, 195 plotting with, 335 unit-object conversion, 193 syntax, 187 UNITS Catalog menu, 187, 188 viewing in the EquationWriter user-defined units, 205 using unit objects in algebraics, application, 240 unit vector, 176 191 for complex numbers, 166 UNITS Catalog menu, 187, 188, Units application, 24, 185-206 arithmetic operations, 203 UNITS Command menu, 187 assembling unit objects, 206 units of angle, 198 building unit objects using the unused memory (free memory), EquationWriter application, 204 up one directory, 122 built-in units, 193 upper tail probabilities, 384 comparing unit objects, 202 uppercase letters, 50, 52 conversion to SI base units, 196 user flags, 515 creating unit objects, 188 user flags annunciator, 48 creating unit objects in the user keyboard, 216 command line, 190 clearing key assignments, 219 dimensionless units of angle, customizing operations, 220 198 editing key assignments, 220 making key assignments, 217 disassembling unit objects, 206 entering and editing unit reactivating a key, 219 user keyboard active annunciator, objects, 188 factoring expressions, 199 ideal gas equation, 185 user memory, 100 International System of Units User mode, 216, 223 user-defined derivatives, 422 (SI), 187 percent calculations, 202 are prefixed by "der", 108 photometric units, 198 user-defined errors, 546 powers of ten prefixes, 192 user-defined functions, 150-155 precedence of functions, 191 are actually programs, 154 prefixing user-defined units, 206 compared to built-in functions, raising a unit object to a power, 150 201 creating, 151 reciprocal of a unit object, 201 executing, 152 reviewing unit names, 191 nesting, 153 temperature conversion, 197 plotting, 334 trigonometric operations, 203 user-defined menus, 213 unit conversion, 188 user-defined transformations, 414 unit object arithmetic, 200 user-defined units, 205

user-key assignments, lost after recovering memory, 102 user-unit coordinates in plots, 323

#### V

value of symbolic constants, 144 VAR menu, 106, 112, 118 reordering, 113 variable, menu labels give name, variables, 105-117 are named storage locations, arithmetic with, 115 can store directories, 118 changing the contents of a variable, 111 common variables, 105 containing a directory object, 123 creating, 106, 107 defining, 107 duplicate names, 121 error recovery from accidentally purging, 115 evaluating a variable's name, evaluating variables containing programs, 110 global variables, 105 immediate execution, 112 in custom menus, 213 in other directories, 121 local variables, 105 memory used by, 101 menu labels, 106 names, 84, 108 new variables are added to the current directory, 121 printing, 606 purging, 114

purging all variables in a directory, 115 recalling contents, 110 reordering the VAR menu, 113 reserved variables, 108 returning object type number of object stored in a variable, 97 Review Catalog, 112 scope of local variables, 105 searching for variable name during evaluation, 121 separating variable names by object type, 98 stored in variables, 394 storing, 107 temporary variables, 105 that contain directories, 110 that contain names, 110 using global variables, 106 using its contents, 109 using quoted versus unquoted variable names, 112 viewing and editing, 67 vectors, 83, 170 - 185 absolute value, 176 are arrays, 345 arithmetic with, 353 arithmetic with matrices, 355 assembling, 173, 183 calculations, 176 commands, 183 compared to complex numbers, 166, 167, 184 complex, 357 cross product, 176, 353 disassembling, 173, 183 display modes, 350 dividing into a matrix, 355 dot product, 176, 353 getting the n-th vector element, how they are displayed, 170

internal representation, 171
keying in, 172
length, 353
put replaces *n-th* element, 90
unit vector, 176
VECTR menu, 142, 171, 183
viewing stack contents, 71

#### W

wait suspending program execution, using the argument 0, 539 Warranty, 673 where function, 416 in the EquationWriter application, 236 WHILE...REPEAT...END, 511 wildcards, with backup objects, Wire Transmission mode, 617 word, certain operations use the concept of, 68 wordsize, binary integers, 207 write-protect switch in RAM cards, 641 installing plug-in cards, 636

### X

x-axis display range, specifying,
295

XLIB names
are objects provided by plug-in
cards, 89
object type number, 97

XON/XOFF handshaking, during
printing, 610

XON/XOFF pacing, 619
xor
with binary integers, 210
with tests, 493

χ<sup>2</sup> test, 384 x-th root, in the EquationWriter application, 232

#### Y

y-axis display range, specifying, 295

#### Z

ZOOM menu, 305 zoom operations, 304 zoom-to-box, 306 zoom-to-box with autoscaling, 306

#### **Contacting Hewlett-Packard**

For Information About Using the Calculator. If you have questions about how to use the calculator, first check the table of contents, the index, and "Answers to Common Questions" in appendix A. If you can't find an answer in the manual, you can contact the Calculator Support department:

Hewlett-Packard Calculator Support 1000 N.E. Circle Blvd. Corvallis, OR 97330, U.S.A.

(503) 757-2004 8:00 a.m. to 3:00 p.m. Pacific time Monday through Friday

For Service. If your calculator doesn't seem to work properly, refer to appendix A for diagnostic instructions and information on obtaining service. If you are in the United States and your calculator requires service, mail it to the Corvallis Service Center:

Hewlett-Packard Corvallis Service Center 1030 N.E. Circle Blvd. Corvallis, OR 97330, U.S.A. (503) 757-2002

If you are outside the United States, refer to appendix A for information on locating the nearest service center.

**HP Calculator Bulletin Board System.** The Bulletin Board provides for the exchange of software and information between HP calculator users, developers, and distributors. It operates at 300/1200/2400 baud, full duplex, no parity, 8 bits, 1 stop bit. The telephone number is (503) 750-4448. The Bulletin Board is a free service—you pay for only the long-distance telephone charge.

## **Contents**

## Part 4: Programming

Page 468 25: Programming Fundamentals

488 26: Tests and Conditional Structures

501 27: Loop Structures

515 28: Flags

519 29: Interactive Programs

541 30: Error Trapping

547 31: More Programming Examples

# Part 5: Printing, Data Transfer, and Plug-Ins

602 32: Printing

612 33: Transferring Data to and from the HP 48

635 34: Using Plug-in Cards and Libraries

# **Appendixes and Indexes**

656 A: Support, Batteries, and Service

677 B: Messages

694 C: HP 48 Character Codes

697 D: Menu Numbers and Menu Maps 699 E: Listing of HP 48 System Flags

707 Operation Index

823 Index



Reorder Number 00048-90003

00048-90078 English Printed in Canada 7/90